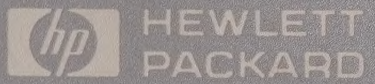

HEWLETT
PACKARD

HEWLETT PACKARD

HP 4954A
Protocol
Analyzer

Library
Reference

HP 18320A
DataCommC Programming Language

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_05515

HP 4954A Protocol Analyzer

HP 18320A DataCommC Programming Language

Library Reference

Manual Part Number: 18320-99503
Microfiche Part Number: 18320-98805

Printed in U.S.A. February, 1989
E0289

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

© Copyright 1989 Hewlett-Packard Company.
Colorado Telecommunications Division
5070 Centennial Boulevard
Colorado Springs, CO, 80919-2497

Contents

1. Programming Conventions

Introduction 1-1
Path and File Names 1-1
Case Sensitivity 1-1
File Extension 1-2
Viewing Program Results 1-2

2. Library Overview

Character Classification and Conversion 2-1
Data Control 2-2
Directory Control 2-2
Display 2-3
Events 2-4
File Manipulation 2-4
Formatting 2-6
Front-End Control 2-6
Keyboard 2-8
Lead and Pod Control 2-8
Math 2-9
Memory Allocation 2-10
Messages and Error Handling 2-10
Miscellaneous 2-11
Port 2-11
Process Control 2-12
Real-Time Clock 2-12
Run-Time Display 2-12
String Manipulation 2-13
Timing Services 2-14

3. Library Functions

abs( ) 3-2
acos( ) 3-3
asin( ) 3-4
assert( ) 3-5
atan( ), atan2( ) 3-7
atof( ) 3-9
atoi( ) 3-11
atol( ) 3-13
beep( ) 3-15
bufcpy( ) 3-16
calloc( ) 3-17
ceil( ) 3-19
center( ) 3-20
ch_dir( ) 3-21
clear_timer( ) 3-22
clearn( ) 3-24
close( ) 3-25
cos( ) 3-27
cosh( ) 3-28
cotan( ) 3-29
cursor_off( ) 3-31
cursor_on( ) 3-32
display_keys_on ( ) 3-33
eq_file_time( ) 3-36
error_notification( ) 3-38
examine_data( ) 3-39
exit( ) 3-41
exp( ) 3-43
fabs( ) 3-44
fclose( ) 3-45
fflush( ) 3-47
fgetc( ) 3-49
fgets( ) 3-50
file_time( ) 3-52
filln( ) 3-54
floor( ) 3-55
fopen( ) 3-56
fprintf( ) 3-59
fputc( ) 3-62
fputs( ) 3-64
fread( ) 3-66

free( ) 3-68
fscanf( ) 3-70
fseek( ) 3-74
ftoa( ) 3-76
fwrite( ) 3-78
get_attribute( ) 3-80
get_bitrate( ) 3-83
get_channelconfig( ) 3-84
get_cursor( ) 3-86
get_data_source( ) 3-87
get_datacode( ) 3-89
get_dir( ) 3-91
get_duplex( ) 3-92
get_errorcheck( ) 3-94
get_event_bounds( ) 3-96
get_ignored_leads( ) 3-99
get_lead( ) 3-101
get_lead_control( ) 3-104
get_mode( ) 3-106
get_parity( ) 3-108
get_protocol( ) 3-110
get_resyncafter( ) 3-112
get_resynchars( ) 3-114
get_resyncmode( ) 3-116
get_startbcc( ) 3-118
get_stop_states( ) 3-120
get_stopbcc( ) 3-122
get_synchars( ) 3-124
get_time( ) 3-126
get_transtext( ) 3-128
getc( ),getchar( ) 3-130
getch( ) 3-132
gets( ) 3-134
hold_event( ) 3-136
init_rs232( ) 3-138
init_trigger 3-139
is_key_avail( ) 3-141
is_msg_avail( ) 3-142
isalnum( ) 3-144
isalpha( ) 3-146
isascii( ) 3-147
iscntrl( ) 3-148

isdigit( ) 3-149
isgraph( ) 3-150
islower( ) 3-151
isprint( ) 3-152
ispunct( ) 3-153
isspace( ) 3-154
isupper( ) 3-155
isxdigit( ) 3-156
itoa( ) 3-157
keyboard_lock( ) 3-159
keyboard_unlock( ) 3-161
log( ) 3-162
log10( ) 3-163
longjmp 3-164
lseek( ) 3-166
ltoa( ) 3-168
make_file( ) 3-170
makedir 3-171
malloc( ) 3-173
open( ) 3-175
parsedir( ) 3-178
pow( ) 3-180
print_char( ) 3-181
printf( ) 3-182
put_all_sks( ) 3-185
put_sk( ) 3-187
putc( ) 3-189
putch( ) 3-191
puts( ) 3-192
rand( ) 3-193
read( ) 3-194
read_message( ) 3-196
read_pod_id( ) 3-198
read_rs232( ) 3-200
read_vidram( ) 3-201
release_event( ) 3-203
resend_bops( ) 3-205
reset_pod( ) 3-208
restore_cursor( ) 3-209
resync( ) 3-210
roll_down( ) 3-212
roll_up( ) 3-213

rs232_lock 3-214
rs232_read_ready( ) 3-215
rs232_unlock( ) 3-217
rs232_write_ready( ) 3-218
save_cursor( ) 3-220
scanf( ) 3-221
send_message( ) 3-225
sendf( ) 3-228
set_attribute 3-233
set_bcc( ) 3-236
set_bitrate( ) 3-238
set_buffer_sizes( ) 3-240
set_channelconfig( ) 3-242
set_cursor( ) 3-245
set_data_source( ) 3-247
set_disp_bank( ) 3-249
set_duplex( ) 3-251
set_error_handler( ) 3-253
set_ignored_leads( ) 3-255
set_lead( ) 3-257
set_lead_control( ) 3-260
set_protocol( ) 3-262
set_resync( ) 3-269
set_rows( ) 3-272
set_screen_mode( ) 3-273
set_stop_states( ) 3-274
set_sync( ) 3-276
set_time( ) 3-278
set_timer( ) 3-280
set_transtext( ) 3-282
setjmp( ) 3-284
sin( ) 3-286
sinh( ) 3-287
spawn( ) 3-288
spawnw( ) 3-291
sprintf( ) 3-294
sqrt( ) 3-297
srand( ) 3-298
sscanf( ) 3-299
start_data( ) 3-303
start_display( ) 3-305
stop_data( ) 3-309

stop_display( ) 3-311
strcat( ) 3-313
strchr( ) 3-314
strcmp( ) 3-315
strcmpi( ) 3-317
strcpy( ) 3-319
strlen( ) 3-320
strncat( ) 3-321
strncmp( ) 3-323
strncpy( ) 3-325
strrchr( ) 3-327
strsave( ) 3-329
strtrm 3-330
sys_msg( ) 3-331
tan( ) 3-332
tanh( ) 3-333
toascii( ) 3-334
tolower( ) 3-336
toupper( ) 3-338
trigger_on_message( ) 3-340
ungetc( ) 3-342
ungetch( ) 3-344
unlink( ) 3-346
wait( ) 3-348
wait_data( ) 3-350
wait_lead( ) 3-353
wipe( ) 3-356
write( ) 3-357
write_rs232( ) 3-359
write_vidram( ) 3-361

4. Include Files

ascii.include 4-2
assert.include 4-3
conio.include 4-3
ctype.include 4-5
decode.include 4-6
dlib.include 4-6
ebcdic.include 4-9
fcntl.include 4-10
leads.include 4-10
math.include 4-13
message.include 4-13
retval.include 4-22
setjmp.include 4-22
stdio.include 4-23
stdlib.include 4-24
string.include 4-25
system.include 4-25
time.include 4-26
video.include 4-26

A. Video Character Sets

Graphics Display Characters A-2
ASCII, Hex and EBCDIC Display Characters A-6

Printing History

New editions are complete revisions of the manual. Update packages (formerly known as "Manual Changes") are issued between editions. They contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change when an update is incorporated.

Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correlation between product updates and manual updates.

Edition 1 ..... February 1989

1

Programming Conventions

Introduction

This manual describes the DataCommC functions included in the DataCommC libraries. It assumes that you are familiar with the C language and with the DataCommC Development and Run-time Environments. If you have questions about editing, compiling, linking or running a program, see *The DataCommC Programming Language User's Guide*. If you have a question about the C language, see *The C Programming Language* by Kernighan and Ritchie, which is included in the documentation package.

Path and File Names

Some of the functions in the DataCommC library expect to receive strings representing path and file names as input arguments. These functions break the path/file name string into its individual components (drive:directory/filename.extension) and then pass the components to the HP 4954A Protocol Analyzer's operating system. The operating system controls the directory/file structure of the protocol analyzer's disc system.

Case Sensitivity

The DataCommC Programming Language and the protocol analyzer's operating system are both case sensitive to the file name preceding the '.' and extension name, so each of the following file names refers to a different file:

```
FILE1.csource

file1.csource

File1.csource
```

File Extensions

In general, all references to file names must use the file's complete extension name. The include files used with the examples in this manual are all shown as `.include` files; however, they could have been entered with `.h` extensions. This is one of the two cases in which the full DataCommC file extension does not need to be used. In the DataCommC environment:

this extension is equivalent to	this one.
`.include`	`.h`
`.csource`	`.c`

See *The DataCommC Programming Language User's Guide* for more information about the other DataCommC file extensions.

Viewing Program Results

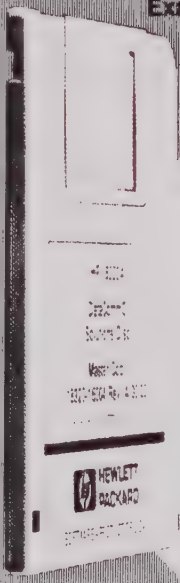
After a DataCommC program has finished executing, control is passed back to the DataCommC environment, and any information the program placed on the display is lost as it is overwritten by the Run Program Menu. This can be avoided if the last routine in the program is an endless loop (for example, `while(1);`) or a wait-for-input (`getch();`) function. Pause functions also work, but are a little more difficult to implement.

Here is a sample program showing the wait-for-input function at the end:

```
main()
{
   printf("Hello, world",\n);

   getch();
}
```

In this case, the text `Hello, world`, remains on the display until a key is pressed on the keyboard. If `getch()` is replaced with `while(1)`, the program waits until the Reset key is pressed before terminating.



Example:

```
#include <message.include>
#include <retval.include>

main()
{
    MESSAGE msg;
    int result;

    if((result = start_data()) != SUCCESSFUL)
       {
       printf("Data Transmit/Receive cannot be started - error # %d.", result);
       exit();
       }
    read_message(&msg, WAIT, WAIT_FOREVER);
    stop_data();
    printf("The message type read is: %d\n", msg.type);
    getch();
}
```

2

Library Overview

Character Classification and Conversion

isalnum( )	Tests for alphanumeric character.
isalpha( )	Tests for alphabetic character.
isascii( )	Tests for ascii character.
iscntrl( )	Tests for control character.
isdigit( )	Tests for decimal digit.
isgraph( )	Tests for printable/whitespace character.
islower( )	Tests for lowercase character.
isprint( )	Tests for printable character.
ispunct( )	Tests for punctuation character.
isspace( )	Tests for whitespace character.
isupper( )	Tests for uppercase character.
isxdigit( )	Tests for hex digit.
toascii( )	Converts a character to ascii.
tolower( )	Converts character to lowercase (if possible).
toupper( )	Converts character to uppercase (if possible).

Data Control

get_data_source( )	Gets the current data source and data files values.
resend_bops( )	Retransmits the last bit-oriented protocol string sent on the line.
sendf( )	Sends a formatted string to the line.
set_buffer_sizes( )	Sets the sizes for the primary DTE and DCE data buffers.
set_data_source( )	Allows a program to set either the line or a disc file as the source for the data buffer input.
start_data( )	Activates the data link control hardware and starts data acquisition.
stop_data( )	Halts the data link control hardware and stops data acquisition.
wait_data( )	Gets the first message of type DATACOMM.

Directory Control

ch_dir( )	Sets the current directory.
get_dir( )	Gets the current directory.
makedir( )	Creates a string representing a directory path.
parsedir( )	Breaks the given directory path into its components.

Display

center( )	Writes a string centered on the display.
cursor_off( )	Makes the cursor invisible.
cursor_on( )	Makes the cursor visible.
get_attribute( )	Returns the value of the current screen attribute.
get_cursor( )	Returns the current row and column position of the cursor.
put_all_sks( )	Writes the given labels onto the corresponding softkeys.
put_sk( )	Writes a label onto one of the eight softkeys.
putch( )	Writes a character to the display at the current cursor position.
puts( )	Writes a string to the display.
read_vidram( )	Reads the video RAM character and attribute value at the current cursor location.
restore_cursor( )	Restores the cursor to the display position.
roll_down( )	Rolls lines of text toward the bottom of the display.
roll_up( )	Rolls lines of text toward the top of the display.
save_cursor( )	Saves the current position of the cursor.
set_attribute( )	Sets the value of the current display attribute.
set_cursor( )	Places the cursor on the row and column indicated.
set_disp_bank( )	Selects one of the four graphics character display banks.
set_rows( )	Defines a text window on the display.

set_screen_mode( )	Sets the screen mode to normal or tranparent text mode.
sys_msg( )	Writes the given string to the display.
wipe( )	Clears the display.
write_vidram( )	Writes a character and its attribute to the video RAM location.

Events

get_event_bounds( )	Gets pointers to the event buffer's first and last events.
hold_event( )	Retains access to the event specified.
release_event( )	Releases access to a specified event in the event buffer.

File Manipulation

close( )	Closes a file opened by the open function.
eq_file_time( )	Compares two files to determine which was modified more recently.
fclose( )	Closes a file opened by the fopen function.
fflush( )	Flushes the buffer associated with a file.
fgetc( )	Gets a character from the specified file.
fgets( )	Gets a string from the specified file.
file_time( )	Returns the time when the specified file was last modified.
fopen( )	Opens a file and returns a pointer to the file.
fprintf( )	Prints formatted information into the specified file.

fputc( )	Adds a character to the specified file.
fputs( )	Puts a string into the specified file.
fread( )	Reads an array from the specified file.
fscanf( )	Reads formatted information from the specified file.
fseek( )	Moves the file pointer to a specified location in the file.
fwrite( )	Writes an array to a file or output string.
lseek( )	Moves the file pointer to a specified location in the file.
make_file( )	Creates a file.
open( )	Opens a file.
putc( )	Writes a character to a file or an output stream.
putchar( )	Writes a character to the standard output stream.
read( )	Reads from a file.
ungetc( )	Pushes a character back onto the input stream.
ungetch( )	Pushes a keystroke back into the system queue.
unlink( )	Deletes a file.
write( )	Writes to a file.

Formatting

atof( )	Converts a string of characters to the double number it represents.
atoi( )	Converts a string of characters to an integer representation.
atol( )	Converts a string of characters to a long integer representation.
ftoa( )	Converts a floating-point or double precision number to an ASCII character string.
itoa( )	Converts an integer to a null-terminated character string.
ltoa( )	Converts a long integer to a null-terminated character string.
printf( )	Outputs arguments to the standard output device.
scanf( )	Reads formatted information from the standard input stream.
sprintf( )	Outputs or prints argument string in memory.
sscanf( )	Reads formatted information from the specified buffer or string.

Front-End Control

get_bitrate( )	Gets the current bitrate value.
get_channelconfig( )	Gets the current configuration for a given channel.
get_datacode( )	Gets the currently selected datacode value.
get_duplex( )	Gets the current duplex value (either half- or full-duplex).
get_errorcheck( )	Getsthe currently selected error check value.
get_mode( )	Gets the currently selected synchronization mode.

get_parity( )	Gets the currently selected parity value.
get_protocol( )	Gets the currently selected protocol value.
get_resyncafter( )	Gets the currently selected resyncafter value.
get_resynchars( )	Gets the values of the currently selected resync characters.
get_resyncmode( )	Gets the currently selected resync mode value.
get_startbcc( )	Gets the currently selected startbcc characters.
get_stopbcc( )	Gets the currently selected stopbcc characters.
get_synchars( )	Gets the values of currently selected sync characters.
get_transtext( )	Gets the value of currently selected transparent text character.
resync( )	Manually reestablishes sync on the line.
set_bcc( )	Sets the protocol rules for calculating the block check character.
set_bitrate( )	Sets the bitrate.
set_channelconfig( )	Sets the configuration of the specified channel.
set_duplex( )	Modifies the duplex value.
set_protocol( )	Sets the protocol, mode, datacode, errorcheck and parity values.
set_resync( )	Defines protocol rules for resynching.
set_sync( )	Sets the values of the sync characters and synchronization mode.
set_transtext( )	Defines the protocol rules for transparent text mode.

Keyboard

getc( )	Reads a character from a file.
getch( )	Reads an unbuffered character from the keyboard.
getchar( )	Reads a character from `stdin`, the standard input stream.
gets( )	Reads a character string from the standard input stream.
is_key_avail( )	Checks the keyboard buffer for available keys.
keyboard_lock( )	Locks the keyboard for exclusive use by the current process.
keyboard_unlock	Unlocks keyboard and allows another process to take control of keyboard.

Lead and Pod Control

get_ignored_leads( )	Gets the currently selected ignored lead values.
get_lead( )	Verifies the state of the lead at the top of the incoming message queue.
get_lead_control( )	Gets the current lead control value.
get_stop_states( )	Gets the current stop state for the specified lead.
read_pod_id( )	Reads the pod identification.
reset_pod( )	Resets the attached pod.
set_ignored_leads( )	Sets the ignored leads for the attached pod.
set_lead( )	Sets a lead to the state specified.
set_lead_control( )	Modifies the lead control value.

set_stop_states( )	Sets the stop states of the specified interface lead.
wait_lead( )	Scans the message queue and collects the first message of type DATACOMM.

Math

abs( )	Calculates the absolute value of a value.
acos( )	Calculates the arccosine of a value.
asin( )	Calculates the arcsine of a value.
atan( )	Calculates the arctangent of a value.
atan2( )	Calculates the arctangent when two values are supplied.
ceil( )	Returns the next largest integer.
cos( )	Calculates the cosine of a value.
cosh( )	Calculates the hyperbolic cosine of a value.
cotan( )	Calculates the cotangent of a value.
exp( )	Calculates the exponential function.
fabs( )	Calculates the absolute value for a floating-point number.
floor( )	Returns the next smallest integer.
log( )	Calculates the natural log.
log10( )	Calculates the base 10 log.
pow( )	Calculates a value raised to a power.
rand( )	Returns a positive pseudo-random number between 0 and 32767.

sin( )	Calculates the sine of a value.
sinh( )	Calculates the hyperbolic sine of a value.
sqrt( )	Calculates the square root.
srand( )	Sets the seed of the random-number generator.
tan( )	Calculates the tangent of a value.
tanh( )	Calculates the hyperbolic tangent of a value.

Memory Allocation

calloc( )	Allocates a specified amount of zeroed storage.
free( )	Returns storage to the heap.
malloc( )	Allocates a specified amount of storage.

Messages and Error Handling

error_notification( )	Sends an error message to the error message queue.
init_trigger( )	Initializes the system trigger function.
is_msg_avail( )	Polls the process message queue for an available message.
read_message( )	Reads a message from the message queue.
send_message( )	Sends a message to the specified message queue.
set_error_handler( )	Designates an error message queue.
trigger_on_message( )	Returns when a message set up as a trigger is found..

Miscellaneous

assert( )	Macro used in identifying logic errors.
beep( )	Produces an audible alarm.
longjmp( )	Restores stack environment values.
setjmp( )	Saves the stack environment values.

Port

init_rs232( )	Initializes the serial port.
print_char( )	Prints a character through the RS232 port.
read_rs232( )	Reads a character from the serial port.
rs232_lock( )	Locks the RS232 port for exclusive use by the calling process.
rs232_read_ready	Checks the status of the RS232 receive register.
rs232_unlock	Releases control of the RS232 port.
rs232_write_ready	Checks the status of the RS232 transmit register.
write_rs232( )	Writes a character to the serial port.

Process Control

exit( )	Closes files and exits a process.
spawn( )	Creates a process that can execute independently of the calling process.
spawnw( )	Creates an independent process; the calling process is blocked until the new process terminates.

Real-Time Clock

get_time( )	Captures the current system time from the real-time clock.
set_time( )	Sets the current system time.

Run-Time Display

display_keys_on( )	Sends a message to the Run-time Display process telling to put it in the keyboard-on state.
examine_data( )	Allows the DataCommC examine data process to be spawned programmatically.
start_display( )	Initializes and spawns a Run-time Display or decode process.
stop_display( )	Terminates execution of the active Run-time Display process.

String Manipulation

bufcpy( )	Copies the contents of memory from one location to another.
clearn( )	Clears a specified amount of storage space.
filln( )	Fills a specified amount of storage space with a given character.
strcat( )	Appends (concatenates) one string to the end of another string.
strchr( )	Returns the position of a character in a string.
strcmp( )	Compares two strings.
strcmpi( )	Compares two strings, but ignores case differences.
strcpy( )	Copies one string to another.
strlen( )	Finds the length of a string.
strncat( )	Appends (concatenates) a specified number of characters from one string onto the end of another string.
strncmp( )	Compares a specified number of characters of one string to another string.
strncpy( )	Copies a specified number of characters from one string to another string.
strrchr( )	Returns a pointer to the last occurrence of a character in string.
strsave( )	Returns a pointer to the new copy of the string.
strtrm( )	Trims trailing spaces from a string.

Timing Services

clear_timer( )	Cancels a timer.
set_timer( )	Sets one of the DataCommC timers for the duration specified.
wait( )	Pauses for a given amount of time.

3

Library Functions

This chapter is an alphabetical list of the functions provided with the HP 18320A DataCommC Programming Language. For a list of these functions grouped by operational category, see chapter 2 of this manual, "Library Overview." For more information about the contents of the supplied include files, refer to chapter 4, "Include Files."

In this chapter, each function is described using the following format:

Format:	A program fragment showing the function's declaration type, and the position and declaration types of its parameters, if any. The fragment also lists any necessary include files. If the functions' return values may be represented by the constant values in the `retval.include` file, that file is also listed.
Description:	A description of what the function does, including a list of the acceptable values that may be used for input parameters and the possible values returned by output parameters.
Return Values:	A list of the values that the function may return, and short descriptions of the conditions that could cause those values to be returned.
See Also:	Other related functions in the DataCommC Library.
Example:	A short program showing how the function is used.

Note The examples in this section are provided for clarification only. HP assumes no responsibility for their functionality or fitness for a specific purpose.

abs( )

Format:

```
int abs(num)
int num;
```

Description:

Calculates the absolute value of an integer (num).

Return Values:

Returns the absolute value of num.

See Also:

```
fabs()
```

Example:

```
main()
{
    int num1 = -8, num2;
    num2 = abs(num1);
    printf("The absolute value of %d is %d",num1, num2);
    getch();
}
```

acos( )

Format:

```
#include <math.include>

double acos(x)
double x;
```

Description:

Calculates the arccosine of x where x is a radian value between -1 and 1.

Return Values:

Returns the arccosine of x (in the range 0 to pi).

See Also:

asin(), atan(), atan2(), cos(), cosh(), cotan(), sin(), sinh(), tan(), tanh()

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double radx, degx, x=0.5;

   radx = acos(x);
   degx = radx * 180 / PI;       /* convert to degrees */
   printf("The arccosine of %f is %f degrees, or %f in radians.", x, degx, radx);
   getch();
}
```

asin( )

Format:

```
#include <math.include>

double asin(x)
double x;
```

Description:

Calculates the arcsine of x where x is a radian value between -1 and 1.

Return Values:

Returns the arcsine of x (in the range -pi/2 to pi/2).

See Also:

```
acos(), atan(), atan2(), cos(), cosh(), cotan(), sin(), sinh(), tan(), tanh()
```

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double radx, degx, x=0.5;

   radx = asin(x);
   degx = radx * 180 / PI;       /* convert to degrees */
   printf("The arcsine of %f is %f (degrees), or %f (in radians)", x, degx, radx);
   getch();
}
```

assert( )

Format:

```
#include <assert.include>

assert(expr)
int expr;
```

Description:

`assert()` is a macro useful for identifying logic errors when strategically placed in a program. As the program executes, the assert expression, `expr`, is evaluated, and if the integer result is `FALSE` (zero), a diagnostic message is printed:

```
Assertion failed: expr, file ffff.csource, line lnnn
```

where `ffff` is the name of the source file, including the path, and `nnn` is the line number of the assert statement that failed. The macro then calls `error_notification()` to place an error message on the screen along with softkeys that may be used to choose whether or not program execution continues.

The `assert.include` file defines this macro, and must be included in each program that uses `assert()`. When the assert statement is no longer needed, place the directive `#define NDEBUG` before the `#include <assert.include>` directive, and `assert` functions are excluded from the source file created by the preprocessor.

See Also:

```
error_notification()
```

assert( )

Example:

```
#include <assert.include>

main()
{
   char ch;

   ch = getch();
   assert(ch >= '0' && ch <= '9');
   printf("The result is %c\n", ch);

   getch();
}
```

atan( ), atan2( )

Format:

```
#include <math.include>

double atan(x)
double x;

double atan2(x,y)
double x,y;
```

Description:

The `atan` function calculates the arctangent of `x` where `x` is a radian value between -1 and 1.

The `atan2` function calculates the arctangent of `x/y` where `x` and `y` are each radian values between -1 and 1 (`x` and `y` may be given opposite signs).

Return Values:

The `atan` function returns the arctangent result in the range -pi/2 to pi/2. The `atan2` function returns the arctangent result in the range -pi to pi.

See Also:

`acos(), asin(), cos(), cosh(), cotan(), sin(), sinh(), tan(), tanh()`

atan( ), atan2( )

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{

   double radx, degx, x=0.5;

   radx = atan(x);
   degx = radx * 180 / PI;       /* convert to degrees */
   printf("The arctangent of %f is %f (degrees), or %f (in radians)", x, degx, radx);
   getch();
}
```

atof( )

Format:

```
#include <stdlib.include>

double atof(str)
char *str;
```

Description:

The `atof` function converts a string of characters to the double number that it represents. `atof()` recognizes a string of numeric characters that may (but is not required to) contain a decimal point. If desired, the numerals may be preceded by blanks and tabs (which are ignored) and/or an optional sign, and followed by an `e` or `E` character with an optionally signed integer. For example, this string is valid:

```
"   895.9e-2"
```

Return Values:

Returns the result of the conversion, or, if `str` points to an unrecognizable or non-existent string, zero is returned.

See also:

`atoi(), atol(), ftoa()`

atof( )

Example:

```
#include <stdlib.include>

main()
{
    char *str;
    double result;

    str = "  104.543e5";       /* test leading blanks and positive exponent */
    result = atof(str);
    printf("%s was converted to %f\n\n", str, result);
    getch();
}
```

atoi( )

Format:

```
#include <stdlib.include>

int atoi(str)
char *str;
```

Description:

Converts a string of characters to an integer representation. Recognizes a string containing leading blanks and tabs (which are ignored), an optional sign, then a string of digits.

Return Values:

Returns the result of the conversion, or zero if `str` points to an unrecognizable or non-existent string.

See Also:

```
atof(), atol(), ftoa()
```

atoi( )

Example:

```
#include <stdlib.include>

main()
{

   char *str;
   int result;

   str = "  104";      /* test leading blanks */
   result = atoi(str);

   printf("%s was converted to %d\n\n", str, result);
   getch();
}
```

atol( )

Format:

```
#include <stdlib.include>

long atol(str)
char *str;
```

Description:

Converts a string of characters to a long integer representation. The function expects to receive a string containing optional leading blanks and tabs (which are ignored), an optional sign, then a string of digits.

Return Values:

Returns the result of the conversion, or zero if `str` points to an unrecognizable or non-existent string.

See Also:

`atof(), atoi(), ftoa()`

atol( )

Example:

```
#include <stdlib.include>

main()
{

    char *str;
    long result;

    str = "  104500";
    result = atol(str);

    printf("%s was converted to %ld\n\n", str, result);
    getch();
}
```

beep( )

Format:

```
void beep()
```

Description:

Causes the protocol analyzer to generate an audible alarm for 1/2 second.

Return Values:

None

Example:

```
main()
{
   beep();
}
```

bufcpy( )

Format:

```
void bufcpy(dest, source, n)
char *dest;
char *source;
int n;
```

Description:

Copies n bytes from source to dest, regardless of any control characters.

Return Values:

None

Example:

```
main()
{

    char *source;
    char dest[80];

    source = "hello world";
    bufcpy(dest,source,strlen(source));
    dest[strlen(source)] = 0;
    printf("The buffer copied is \"%s\".",dest);
    getch();
}
```

calloc( )

Format:

```
#include <stdlib.include>

char *calloc(num, size)
unsigned int num;
unsigned int size;
```

Description:

Allocates `num` contiguous units (blocks) of zeroed memory of `size` bytes each. The total size of the allocated block (`num` multiplied by `size`) cannot exceed 32K bytes. When memory allocated in this way is no longer being used, it should be returned to the system by calling `free()`.

Return Values:

Returns pointer to the allocated memory, or `NULL` if the operation fails.

See Also:

`free()`, `malloc()`

calloc( )

Example:

```
#include <stdlib.include>

main()
{
   char *str, *new_str;

   str="hello world";
   if(new_str=calloc(1,strlen(str) + 1)) /* add 1 to the length for the NULL char. */
   {
      strcpy(new_str,str);
      printf("The new_str is %s.",new_str);
   }
   else
      printf("Memory could not be allocated");
   getch();
}
```

ceil( )

Format:

```
#include <math.include>

double ceil(x);
double x;
```

Description

Calculates the value (in `double` representation) of the next integer larger than `x`.

Return Values:

Returns the value of the next largest integer.

Example:

```
#include <math.include>

main()
{
   double x = 1.25;
   double x1;

   x1 = ceil(x);
   printf("ceil(%f) is %f.", x, x1);
   getch();
}
```

center( )

Format:

```
void center(string,row)
char *string;
int row;
```

Description:

This function writes a string to the display (padded on either side with spaces to center it) with the current display attributes. This takes the whole row.

The cursor position is moved to row `row`, column 1.

If the row is specified out of range (that is, larger than 25), no action is taken and the cursor is not moved. Similarly, no action is taken if string is `NULL`. If `string` is empty ("") the row is filled with spaces.

Return Values:

None

Example:

```
main()
{

   center("This message is centered on line 10.", 10);
   getch();
}
```

ch_dir( )

Format:

```
void ch_dir(path)
char *path;
```

Description:

Sets the current global directory.

Return Values:

None

See Also:

```
get_dir(), makedir(), parsedir()
```

Example:

```
main()
{
   char new_path[81];

   ch_dir("c:C");

   get_dir(new_path);
   printf("The current default path is %s.",new_path);
   getch();
}
```

clear_timer( )

Format:

```
int clear_timer(timer_number)
int timer_number;
```

Description:

Cancels a timer, using `timer_number` to identify the selected timer.

Return Values:

Returns the number of the cancelled timer when successful, or

`-1` if the timer number does not correspond to an active timer; the request is ignored.

See Also:

`set_timer()`

Example:

```
main()
{

   int timer,cleared_timer;

   if((timer = set_timer(5L)) == -1)         /* set the timer for 1/2 second */
      printf("Timer could not be set.");

   else
   {
      printf("timer %d set.\n",timer);
      if((cleared_timer = clear_timer(timer)) != -1) /* try to clear it */
         printf("Timer %d cleared.",cleared_timer);

      else
         printf("Timer %d could not be cleared.",timer);
   }
   getch();
}
```

clearn( )

Format:

```
void clearn(n, ptr)
int n;
char *ptr;
```

Description:

Clears n bytes beginning at ptr.

Return Values:

None

Example:

```
main()
{
   char *str;

   str = "Hello World";

   printf("The string before the clearn() call is %s.",str);
   clearn(strlen(str) + 1,str);
   printf("The string after the clearn() call is %s.",str);
   getch();
}
```

close( )

Format:

```
#include <fcntl.include>

int close(file_desc)
int file_desc;
```

Description:

Closes a file opened by the open function.

Return Values:

0	The close operation was successful.
-1	An error occurred during the close operation.

See Also:

open( )

close( )

Example:

```
#include <fcntl.include>

main()
{

   int fdesc;

   if ((fdesc = open("c:TEMP",O_WRONLY)) == -1)
      printf("Error in file opening!");

   else
      close(fdesc);
   getch();
}
```

cos( )

Format:

```
#include <math.include>

double cos(x)
double x;
```

Description:

Calculates the cosine of x.

Return Values:

Returns the cosine of x (a value between 1 and -1).

See Also:

acos(), asin(), atan(), atan2(), cosh(), cotan(), sin(), sinh(), tan(), tanh()

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{

   double y, x=PI/2;

   y = cos(x);
   printf("The cosine of %f is %f.", x, y);
   getch();
}
```

cosh( )

Format:

```
#include <math.include>

double cosh(x)
double x;
```

Description:

Calculates the hyperbolic cosine of x.

Return Values:

Returns the hyperbolic cosine of x (a value between 1 and -1).

See Also:

```
acos(), asin(), atan(), atan2(), cos(), cotan(), sin(), sinh(), tan(), tanh()
```

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{

   double y, x=PI/2;

   y = cosh(x);
   printf("The hyperbolic cosine of %f is %f.", x, y);
   getch();
}
```

cotan( )

Format:

```
#include <math.include>

double cotan(x)
double x;
```

Description:

Calculates the cotangent of x.

Return Values:

Returns the cotangent of x.

See Also:

acos(), asin(), atan(), atan2(), cos(), cosh(), sin(), sinh(), tan(), tanh()

cotan( )

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double radx, degx, x=0.5;

   radx = cotan(x);
   degx = radx * 180/PI;

   printf("The cotangent of %f is %f degrees or %f in radians.", x, degx,radx);
   getch();
}
```

cursor_off( )

Format:

```
void cursor_off()
```

Description:

Makes the cursor invisible.

Return Values:

None

See Also:

```
cursor_on()
```

Example:

```
main()
{
   cursor_on();
   printf("Press a key: ");
   getch();
   printf("\n");
   cursor_off();
   printf("Press a key: \n");
   getch();
}
```

cursor_on( )

Format:

```
void cursor_on()
```

Description:

Makes the cursor visible.

Return Values:

None

See Also:

```
cursor_off()
```

Example:

```
main()
{
    cursor_off();
    printf("Press a key: \n");
    getch();
    cursor_on();
    printf("Press a key: ");
    getch();
}
```

display_keys_on( )

Format:

```
int display_keys_on()
```

Description:

Sends a message to the display telling it to put itself in the keyboard-on state. The process which is sharing the keyboard with the display should have one softkey labeled **Display Control** , and if that softkey is pressed, the process should: 1) unlock the keyboard; 2) call `display_keys_on()`; and 3) refrain from painting any softkeys or getting keys until it receives a `KEYS_OFF` (`DISPLAY` type) message from the display (see `start_display()` for more information on `DISPLAY` messages).

Note

This function is only needed when a display process has been activated (using `start_display()`) with the `share_keyboard` parameter equal to 1. In this case, the display and the process that called `start_display()` may take turns controlling the softkey labels and the `stdin` stream.

The display takes over keyboard control when the process calls `display_keys_on()`, and it gives up control by sending a message of type `DISPLAY` and subtype `STOPPED` to the parent process (the `STOPPED` message may be triggered by pressing the top-level **Stop Run** key, which the display process generates).

Return Values:

0	The message was sent.
-1	The queue has been deleted
-2	An invalid queue number was given.

display_keys_on( )

-3 Out of memory.

-5 Out of memory.

-6 The queue was full.

See Also:

start_display(), stop_display()

Example:

```
#include <stdio.include>    /* NULL                          */
#include <message.include>  /* MESSAGE, WAIT, WAIT_FOREVER */
#include <video.include>    /* NORMAL, INVERSE               */
#include <conio.include>    /* SK1-8                         */

main()
{
   MESSAGE message;
   int result, chr, my_keys_on;

   set_rows(17,21);
   keyboard_lock();
   result = start_display(NULL, 15, 1);
   printf("start_display() returned: %d \n",result);
   put_all_sks("","","","","","Display'Control","","'Exit",INVERSE);
   while(1)
   {
      chr = getch();
      switch(chr)
      {
```

```
        case SK6:  /* Display Control Key*/
            keyboard_unlock();
            result = display_keys_on();
            printf("display_keys_on() returned %d\n",result);
            my_keys_on = 0;
            while (!my_keys_on)
            {
                read_message(&message, WAIT, WAIT_FOREVER);
                printf("message type/subtype: %d / %d", message.type, message.subtype);
                if((message.type == DISPLAY) && (message.subtype == KEYBOARD_OFF))
                {
                    my_keys_on = 1;
                    keyboard_lock();
                    set_rows(17,25);
                    wipe(22,25);
                    set_rows(17,21);
                    put_all_sks("","","","","","Display Control","","Exit",INVERSE);
                }
            }
            break;
        case SK8:  /* Exit Key*/
            stop_display();
            exit();
        default:
            break;
        }
    }
}
```

eq_file_time( )

Format:

```
int eq_file_time(file_1, file_2)
char *file_1;
char *file_2;
```

Description:

This function compares two files to determine which was modified more recently.

Return Values:

-2	An error occurred during an attempt to open one of the files.
-1	The modify time of `file_1` < modify time of `file_2`
0	The modify time of `file_1` = modify time of `file_2`
1	The modify time of `file_1` > modify time of `file_2`

Example:

```
main()
{

   int result;

   result = eq_file_time("file1.text","file2.text");

   switch(result)
   {
      case -2:
         printf("all of the necessary files cannot be found");
         break;
      case -1:
         printf("file1.text is older than file2.text");
         break;
      case 0:
         printf("the files have the same time stamp!");
         break;
      case 1:
         printf("file2.text is older than file1.text");
         break;
   }
   getch();
}
```

error_notification( )

Format:

```
#include <message.include>

void error_notification(yelp_msg_ptr, error_subtype)
char *yelp_msg_ptr, error_subtype;
```

Description:

Sends an error message to the `error_QID`. There are two error subtypes: `FATAL_ERROR` and `NON_FATAL_ERROR`.

Return Values:

None

See Also:

```
set_error_handler()
```

Example:

```
#include <message.include>

main()
{

   error_notification("Out of Memory",NON_FATAL_ERROR);
}
```

examine_data( )

Format:

```
#include <system.include>

int examine_data(path_name, priority)
char *path_name;
int priority;
```

Description:

Allows the DataCommC examine data process to be spawned programmatically. The process does not share the keyboard with other processes, and it uses the entire screen for its display. Once running, the examine data process can only be terminated when the top-level **Exit** softkey, which it generates, is pressed.

If the `path_name` parameter is `NULL`, the default examine data process is used (the default program is: `c:C/System/Bin/ExamEvents.program`). `NULL` is defined in the `stdio.include` file.

The examine data process should be given a priority number between 1 and 199. If the priority is not within this range, `priority` is set to `DEFAULT_PRIORITY`, which is defined (in `system.include`) to be 100.

Return Values:

0	The process was spawned.
-1	The program was not found.
-2	There was no memory for the process.
-3	Invalid program file format found.
-15 to -19	Can't communicate with ROOT process.
-21	No message queues were available.

examine_data( )

-22	Can't notify process manager.
-41	Too many processes.
-42	No memory for user stack.
-43	User stack was too small.

See Also:

start_display(), spawn()

Example:

```
#include <system.include>
#include <stdio.include>

main()
{
    examine_data(NULL, DEFAULT_PRIORITY);
}
```

exit( )

Format:

```
void exit(exit_val)
int exit_val;
```

Description:

Closes files and exits a process. If the process was spawned with `spawnw()`, `exit_val` is returned to the calling process.

Return Values:

None

See Also:

`spawn()`, `spawnw()`

exit( )

Example:

```
main()
{

   int ch;

   printf("Press a key to be echoed to the screen");
   while(1)
   {
      ch = getch();
      if (ch > 31)
         putch(ch);
      else
         exit(1);
   }
}
```

exp( )

Format:

```
#include <math.include>

double exp(x)
double x;
```

Description:

Returns the exponential function of x.

Return Values:

Returns the result of the operation.

See Also:

```
log()
```

Example:

```
#include <math.include>

main()
{
double y, x = 10;

y = exp(x);

printf("The exponential of %f is %f.",x,y);
getch();
}
```

fabs( )

Format:

```
#include <math.include>

double fabs(x)
double x;
```

Description:

Calculates the absolute value of a number represented by a `double` (`x`).

Return Values:

Returns the absolute value of `x`.

See Also:

`abs()`

Example:

```
#include <math.include>

main()
{
   double i,j = -15.25;

   i = fabs(j);

   printf("The absolute value of %f is %f.",j,i);
   getch();
}
```

fclose( )

Format:

```
#include <stdio.include>

int fclose(file)
FILE *file;
```

Description:

Closes a file.

Return Values:

0	The file was closed successfully.
-1	An error occurred during the `fclose` operation.

See Also:

`fopen()`

fclose( )

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;

   printf("Press a key to close testfile.csource: ");
   getch();
   if((fptr = fopen("testfile.csource","w")) != NULL)
   {
      fputs("hello world",fptr);
      fclose(fptr);
      printf("\nThe file was closed.");
   }
   else
      printf("Could not open file.");
   getch();
}
```

fflush( )

Format:

```
#include <stdio.include>

int fflush(file)
FILE *file;
```

Description:

Flushes the buffer associated with a file. The `fflush()` macro is defined in the `stdio.include` file.

Return Values:

`0`	The `fflush` operation was successful.
`-1`	An error occurred during the `fflush` operation.

fflush( )

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;

   printf("Press a key to flush the contents of testfile.csource:");
   if((fptr = fopen("testfile.csource","w")) != NULL)
   {
      fputs("hello world",fptr);
      fflush(fptr);
      fclose(fptr);
      printf("The file contents were flushed.");

   }
   else
      printf("Could not open file.");
   getch();
}
```

fgetc()

Format:

```
#include <stdio.include>

int fgetc(file)
FILE *file;
```

Description:

Gets the next character from a file. This macro is defined in the `stdio.include` file.

Return Values:

Returns the character.

See Also:

`ungetc()`, `fgets()`

Example:

```
#include <stdio.include>

main()
{
   int inchar;

   inchar = fgetc(stdin);
   printf("The character read from the input is %c.", inchar);
   getch();
}
```

fgets( )

Format:

```
#include <stdio.include>

char *fgets(str, n, file)
char *str;
int n;
FILE *file;
```

Description:

Reads up to `n-1` characters from a file or up to the first newline character ('\n'), or until end-of-file, whichever comes first. The string `str` is terminated with a `NULL` character ('\0').

Return Values:

Returns a pointer to the string.

See Also:

`fputs(), fgetc(), ungetc()`

Example:

```
#include <stdio.include>

main()
{
   char instr[70];

   fgets(instr,70,stdin);
   printf("The string read from the input is %s.", instr);
   getch();
}
```

file_time( )

Format:

```
int file_time(name, year, month, day, hour, min, sec)
char *name;
char year[3];
char month[3];
char day[3];
char hour[3];
char min[3];
char sec[3];
```

Description:

Returns the time when the specified file was last modified. The file is specified in the `name` parameter, which must contain the device, path and file name. The `file_time` function returns each of the `year`, `month`, `day`, `hour`, `min` and `sec` parameters as ASCII, null-terminated string.

Return Values:

0	The function was successful.
-1	The file name could not be found.

See Also:

`eq_file_time()`

Example:

```
main()
{
   char year[3], month[3], day[3], hour[3], min[3], sec[3];

   if(file_time("file1.text", year, month, day, hour, min, sec) == 0)
      printf("file1.text was last modified on %s/%s/%s at %s:%s:%s.",
         month, day, year, hour, min, sec);
   else
      printf("Error returned from file time.");
   getch();
}
```

filln( )

Format:

```
void filln(n,ptr,fill)
int n;
char *ptr;
char fill;
```

Description:

Fills `n` bytes with a `fill` character, beginning at `ptr`.

Return Values:

None

Example:

```
main()
{
   char buff[81];

   printf("Enter a string of characters and press RETURN\n");
   gets(buff);
   printf("The buffer contains: %s\n",buff);
   filln(strlen(buff),buff,'/000');    /* initialize the buffer to 0's */
   printf("After filln(), the buffer contains: %s",buff);
   getch();
}
```

floor( )

Format:

```
#include <math.include>

double floor(x)
double x;
```

Description:

Returns the largest integer not greater than x.

Return Values:

Returns the value of the next smaller integer.

See Also:

```
ceil()
```

Example:

```
#include <math.include>

main()
{
double x = 1.25;
double x1;

x1 = floor(x);
printf("floor(%f) is %f.",x,x1);
getch();
}
```

fopen( )

Format:

```
#include <stdio.include>

FILE *fopen(name, access_mode)
char *name;
char *access_mode;
```

Description:

Opens the file specified by the `name` parameter. The `access_mode` parameter gives the type of access requested for the file, as shown below:

access_mode	Description
"r"	Opens the file for reading only. If the file does not exist or cannot be found, an error is returned.
"w"	Opens an empty file for writing. If the given file already exists, its contents are destroyed.
"a"	Opens a file for appending. The calling program is granted write-only access. The current file position is the character after the last character in the file. If the file does not exist, it is created.
"r+"	Opens the file for both reading and writing. If the file does not exist or cannot be found an error is returned.
"w+"	Opens the file for both reading and writing. If the file already exists, its contents are destroyed.
"a+"	Opens a file for both reading and appending. If the file does not exist, it is created.

Note The "w" and "w+" access modes destroy existing files; they should be used only with extreme caution.

Only new files opened using the "w" or "w+" modes are allowed to grow without limit (if disc space allows). All other existing files can only be read or written to the end of the physical file size. Additionally, only one new file may be opened in "w" or "w+" mode at one time on each disc unit. If more than one file must be written to at the same time, all of them, or all but one must have been previously created and must exist prior to calling `fopen()`. Files may be created in this way with `make_file()`.

Return Values:

Returns a pointer to the opened file, or `NULL` if an error occurred.

See Also:

`fclose()`, `fread()`, `fseek()`, `fwrite()`, `open()`

fopen( )

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;

   if((fptr = fopen("testfile.csource","W")) != NULL)
   {
      printf("testfile.csource was opened.");
      fclose(fptr);
   }
   else
      printf("Could not open file.");
   getch();
}
```

fprintf( )

Format:

```
int fprintf(file, format, args)
FILE *file;
char *format;
char *args;
```

Description:

Outputs arguments in `args` to an output stream or file (`file`) according to `format`.

The character string pointed at by `format` directs the output operation, and contains two types of information: ordinary alphanumeric characters, which are output unchanged; and conversion specifications, each of which causes the conversion and output of the next argument in the `args` list.

The formatted string is output from left to right. When a conversion specification is encountered, the next (initially first) argument is output according to the conversion specification.

A conversion specification has the form:

```
%[flag] [width] [.precision] [l] type
```

Each field enclosed in braces "`[ ]`" is optional and consists of a single character or number signifying a particular format option. The simplest possible conversion specification contains a percent sign (`%`) and a conversion character (ex: `%f`).

`flag`

- The converted argument is left-justified when printed (the default is right-justification).

fprintf( )

`width`

digit string The numeric digit string specifies the field width for the conversion. If the converted value has fewer characters than `width`, enough blank (space) characters are output to make the total number of characters output equal the field width. The spaces are output before or after the value, depending on the presence or absence of the left-justification flag. If the field width digits have a leading zero, zeros are used as pad characters instead of spaces.

`*` The `width` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`.precision`

digit string For floating point conversions, `precision` specifies the number of digits to appear after the decimal point; for character string conversions, it specifies the maximum number of characters to be printed from a string.

`*` The `precision` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`l`

A conversion normally performed on an `int` is performed on a `long` (may be used with the `d`, `o` and `x` conversion characters).

The `type` character format is as follows:

character	type of argument	output format
d	int	signed decimal
u	int	unsigned decimal
x	int	unsigned hexadecimal
o	int	unsigned octal
f	float or double	floating point
c	char	single character
s	string	character string
e	float or double	scientific nototation
g	uses d, f or e - whichever gives full precision in minimum space	

Return Values:

Returns the number of characters printed.

See Also:

printf(), sprintf()

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;

   if((fptr = fopen("testfile.csource","w")) != NULL)
   {
      fprintf(fptr,"%s","hello world");
      printf("The string \"hello world\" was placed in testfile.csource.");
      fclose(fptr);
   }
   else
      printf("Could not open file.");
   getch();
}
```

fputc( )

Format:

```
#include <stdio.include>

int fputc(ch, file)
int ch;
FILE *file;
```

Description:

Writes a single character to an output stream or file (`file`). This macro is defined in the `stdio.include` file.

Return Values:

Returns the character written.

See Also:

`fputs(), putc()`

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;
   char *outstr;

   outstr = "hello world";

   while(*outstr)
      fputc(*outstr++,stdout);
   fputc('\n',stdout);
   getch();
}
```

fputs( )

Format:

```
#include <stdio.include>

int fputs(str, file)
char *str;
FILE *file;
```

Description:

Takes the string indicated by the `str` parameter and places all the characters, up to but not including the end `NULL`, on an output stream or in the file described by `file`. If `file` is used to define the output stream as `stdout`, the string is sent to the standard output device (the protocol analyzer's display).

Return Values:

Zero if successful, nonzero if an error has occurred.

See Also:

`fgets(), puts()`

fputs( )

Example:

```
#include <stdio.include>

main()
{
    char *outstr;

    outstr = "hello world";
    fputs(outstr,stdout);
    getch();
}
```

fread( )

Format:

```
#include <stdio.include>

int fread(ptr, size, n, file)
char *ptr;
int size;
int n;
FILE *file;
```

Description:

Reads an array from a file or input stream and places the information in a storage area pointed to by the `ptr` parameter. The `size` and `n` parameters define the size (in bytes) of each array element and the number of elements in the array, respectively. If the `file` parameter indicates a file, and not an input stream, the associated file pointer is incremented.

Return Values:

Returns the number of elements actually read; this number may be smaller than `n` if an error occurred during the transfer or the end-of-file was found before all the elements were read.

See Also:

`fwrite(), gets(), fputs()`

Example:

```
#include <stdio.include>

main()
{
   char instr[70];
   int result;

   result = fread(instr, 1, 10, stdin);
   instr[result] = 0;
   printf("The string read from the input is %s.", instr);
   getch();
}
```

free( )

Format:

```
#include <stdlib.include>        /* for function declarations */

void free(ptr)
char *ptr;
```

Description:

This function deallocates a block of memory previously allocated by `malloc()` or `calloc()`. The size of the freed block is exactly the number of bytes requested in the `malloc/calloc` function call. Similarly, the pointer (`ptr`) passed to `free()` must be identical to the one returned by `malloc()` or `calloc()`.

Return Values:

None

See Also:

`calloc()`, `malloc()`

Example:

```
#include <stdlib.include>

main()
{
   char *str, *allocated_str;

   str="hello world";
   allocated_str=calloc(1,strlen(str) + 1); /* add 1 to the length for the NULL char. */
   strcpy(allocated_str,str);
   printf("The allocated_str is %s.",allocated_str);
   free(allocated_str);                    /* free the memory to the system */
   getch();
}
```

fscanf( )

Format:

```
int fscanf(file, format, args)
FILE *file;
char *format;
char *args;
```

Description:

Takes text characters from the specified stream (`file`), checks the character types against conversion characters imbedded in a control string pointed to by the `format` parameter, and places matching text in the fields pointed to by the `args` list.

The following `fscanf()` example shows the main components of the function:

```
fscanf("file1.text", "%f%s", &fltptr, &strptr);
```

The string `"%f%s"` is a control string (`format`) with two control items (both conversion characters), one indicating a floating point value (`%f`), and the other (`%s`) indicating a string. The other arguments, `&fltptr` and `&strptr`, define the argument list (`args`). If `fscanf()` finds floating point characters, it places them in the memory location pointed to by `fltptr`; if a character string is found next, it is placed in the memory location pointed to by `strptr`.

A control string contains these control items:

- Conversion specifications
- Optional white space characters (tab, space, newline)
- Optional alphanumeric characters (not white space, and not part of a conversion specification).

The `fscanf` function works its way through a control string from left to right, trying to match each control item to a portion of the input stream. During the matching process, `fscanf()` fetches characters one at a time from the input stream. If a character is found which doesn't match the type specifier for the corresponding conversion specificiation, `fscanf()` pushes the character back onto the input stream and finishes processing the current control item. This

"pushing back" frequently gives unexpected results when a stream is used later by other I/O functions, such as `getc()` or `scanf()`, as well as by `fscanf()` itself, if it is used again.

A conversion specification has the form:

```
%[*] [width] [l] type
```

Each field enclosed in braces "`[ ]`" is optional and consists of a single character or number signifying a particular format option. The simplest possible conversion specification contains a percent sign (`%`) and a conversion character (ex: `%f`)

The optional fields are defined below, and conversion characters are discussed later in this segment.

`*`

Assignment suppression character. The current stream is scanned, but not saved. The function goes on to the next control string item.

`width`

This field specifies the maximum number of characters to be fetched for the conversion.

`l`

This field indicates that the argument is a pointer to a long data type - the exact type (for example, long decimal, long hex, long unsigned) is determined by the conversion character.

fscanf( )

The conversion character format is as follows:

character	type of argument	expected input format
d	pointer to int	signed decimal
u	pointer to int	unsigned decimal
x	pointer to int	unsigned hexadecimal
o	pointer to int	unsigned octal
e,f	pointer to float	floating point
c	pointer to char	single character
s	pointer to char array	character string

When a conversion specification is encountered in the control string, the `fscanf` function skips leading white space on the input stream, then collects characters from the stream until it encounters one that is not appropriate for the corresponding conversion character. That character is pushed back onto the input string.

As long as the conversion specification didn't request assignment suppression (see '*', above), the text string that was read from the keyboard is converted to the format specified by the conversion specification, the result is placed in the location pointed to by the corresponding args argument, the next argument becomes current, and the function proceeds to the next control string item.

If assignment suppression was requested, the `fscanf` function ignores the input characters and goes on the the next control item.

If an ordinary character is found in the control string, outside any conversion specfication, `fscanf()` fetches the next character. If that character matches the character in the control string, the function goes on to the next control string item, ignoring the input character. If there is no match, `fscanf()` terminates.

If a white space character is found in the control string, the `fscanf` function fetches input characters until the first non-white space character is read. The non-white space character is pushed back onto the input stream and `fscanf()` proceeds to the next item in the control string.

fscanf( )

Return Values:

Returns the number of items converted and assigned to memory locations in `args`. Unmatched items, since they are not assigned to `args`, are not included in the count. `fscanf()` returns `EOF` if an attempt is made to read past the end of the file.

See Also:

`scanf(), sscanf()`

Example:

```
#include <stdio.include>

main()
{
   char instr[70];

   fscanf(stdin,"%s",instr);
   printf("The string read from the input is %s.", instr);
   getch();
}
```

fseek( )

Format:

```
#include <stdio.include>

int fseek(fptr, offset, origin)
FILE *fptr;
long offset;
int origin;
```

Description:

Moves the file pointer `fptr` to a location in the file `offset` bytes from the `origin`. The `fptr` parameter must be the file pointer returned by `fopen()`. Constant values for `origin` are defined in the `stdio.include` file, and are described below:

origin	Description
SEEK_SET	The file pointer indicates the beginning of the file.
SEEK_CUR	The file pointer indicates the current position in the file.
SEEK_END	The file pointer indicates the end of the file.

Return Values:

Returns zero if the function was successful, or -1 if an error occurred during the operation.

See Also:

`lseek()`

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;
   int open_result, seek_result, write_result;
   char instr[20];

   if((fptr = fopen("file1.text","w")) == NULL)
      printf("Can't open file1.text.");

   else
   {
      write_result = fwrite("hello world\n",1,10,fptr);
      if((seek_result = fseek(fptr,7L,SEEK_SET)) != 0)
         printf("Seek error detected.");
      else
      {
         fclose(fptr);
         if((fptr = fopen("file1.text","r")) == NULL)
            printf("Can't open file1.text.");
         else if ((seek_result = fseek(fptr,7L,SEEK_SET)) != 0)
            printf("Seek error detected.");
         else
         {
            fgets(instr,3,fptr);
            printf("the string read after fseek is %s.",instr);
         }
      }
   }
   getch();
}
```

ftoa( )

Format:

```
#include <stdlib.include>

void ftoa(val,buf,precision,type)
double val;
char *buf;
int precision,type;
```

Description:

Converts a floating-point or double precision number to an ASCII character string.

The `val` parameter is the number to be converted and `buf` points to the buffer where the ASCII string will be placed. The `precision` parameter specifies the number of digits to the right of the decimal point, and `type` specifies the format: 0 for "`E`" format, 1 for "`F`" format, and 2 for "`G`" format.

Return Values:

None

See Also

`atof()`

Example:

```
#include <stdlib.include>

main()
{
    char str[25];
    double dvar;

    dvar = 3.456;
    ftoa(dvar,str,3,2);
    printf("%s ascii was converted from %f.",str,dvar);
    getch();
}
```

fwrite( )

Format:

```
#include <stdio.include>

int fwrite(ptr, size, n, file)
char *ptr;
int size;
int n;
FILE *file;
```

Description:

Writes an array to a file or output stream. The `fwrite` function reads an array from a buffer pointed to by the `ptr` parameter, and places the information in a file or output stream. The `size` and `n` parameters define the size (in bytes) of each array element and the number of elements in the array, respectively. If the `file` parameter indicates a file, and not an output stream, the associated file pointer is incremented.

Return Values:

Returns the number of elements actually written; this number may be smaller than `n` if an error occurred during the transfer.

See Also:

`fread()`

Example:

```
#include <stdio.include>

main()
{
   FILE *fptr;
   char *outstr;

   outstr = "hello world";
   if((fptr = fopen("testfile.csource","W")) != NULL)
   {
      fwrite(outstr,1,strlen(outstr),fptr);
      printf("The string \"hello world\" was written to testfile.csource.");
      fclose(fptr);
   }
   else
      printf("Could not open file.");
   getch();
}
```

get_attribute( )

Format:

```
#include <video.include>

int get_attribute()
```

Description:

Returns the value of the current display attribute. The attribute determines the form of any text printed to the screen, and generates ASCII, hex or EBCDIC character sets in normal, inverse-video, low-intensity or blinking modes. Any of the attributes for a given character set may be arithmetically OR'd together to form a combination of effects.

The UNDERLINE attribute allows any of the characters in the ASCII, hex or EBCDIC character sets to be underlined when they appear on the screen.

Additionally, the SPECIAL attribute allows access to four sets or banks of graphics characters. These individual banks may be selected with the set_disp_bank function after the SPECIAL attribute is active. See set_disp_bank() in this chapter.

The constant values available for the attribute parameter are located in the video.include file and are also shown in this segment.

Return Values:

The get_attribute function may return almost any combination of these constant values, since set_attribute() may arithmetically OR the attributes together. For this reason, it is a good idea to take the return value and arithmetically AND it with the constant value of the attribute you are testing.

get_attribute( )

ASCII attribute	value
NORMAL	0x0200
INVERSE	0x0600
BLINK	0x0A00
HALF_BRIGHT	0x1200

Hex attribute	value
HEX_NORMAL	0x0000
HEX_INVERSE	0x0400
HEX_BLINK	0x0800
HEX_HALF_BRIGHT	0x1000

EBCDIC attribute	value
EBCDIC_NORMAL	0x0100
EBCDIC_INVERSE	0x0500
EBCDIC_BLINK	0x0900
EBCDIC_HALF_BRIGHT	0x1100

Underline attribute	value
UNDERLINE	0x2000

Graphics attribute	value
SPECIAL	0x0300

See Also:

set_attribute(), set_disp_bank(), set_screen_mode()

get_attribute( )

Example:

```
#include <video.include>

main()
{
   int attribute;
   attribute = get_attribute();
   printf("The following screen attributes are set:\n\n");

   if( (attribute &  NORMAL) == NORMAL)
      printf("       NORMAL\n");
   if( (attribute &  SPECIAL) == SPECIAL)
      printf("       SPECIAL\n");
   if( (attribute &  INVERSE) == INVERSE)
      printf("       INVERSE\n");
   if( (attribute &  HALF_BRIGHT) == HALF_BRIGHT)
      printf("       HALF_BRIGHT\n");
   if( (attribute &  BLINK) == BLINK)
      printf("       BLINK\n");
   if ( (attribute & NORMAL) == 0)
   {
      if( attribute &  HEX_NORMAL)
         printf("       HEX_NORMAL\n");
      if( attribute &  HEX_INVERSE )
         printf("       HEX_INVERSE\n");
      if( attribute & HEX_BLINK )
         printf("       HEX_BLINK\n");
   }
   getch();
}
```

get_bitrate( )

Format:

```
#include <retval.include>

int get_bitrate(bitrate)
long *bitrate;
```

Description:

Puts the value of the currently selected bitrate into the memory location pointed to by the `bitrate` parameter.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See Also:

```
set_bitrate()
```

Example:

```
main()
{
   long bitrate;

   get_bitrate(&bitrate);
   printf("The bitrate is %ld.",bitrate);
   getch();
}
```

get_channelconfig( )

Format:

```
#include <dlib.include>
#include <retval.include>

int channelconfig(channel, config)
char channel;
char *config;
```

Description:

Checks the configuration for the channel specified by the `channel` parameter, then puts the information into the memory location pointed to by the `config` parameter. Constant values for both parameters are in the `dlib.include` file; valid values are also shown below:

channel	value
DTE	0x1
DCE	0x2
SDTE	0x4
SDCE	0x5

config	value
NONE	0x0
RX	0x1
TX	0x2

Return Values:

`SUCCESSFUL (0)`

`WARNING_1 (1)` The value specified for `channel` is unknown. It is assumed to be the default, `DCE`, and the `DCE` configuration is passed back to `config`.

See Also:

`set_channelconfig()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   char config;
   int result;

   printf("\n");

   if( (result = get_channelconfig(DTE,&config)) == SUCCESSFUL )
      printf("The DTE channel configuration is %d.",config);
   else
      printf("get_channelconfig() error #%d.",result);
   getch();
}
```

get_cursor( )

Format:

```
void get_cursor(row, column)
int *row, *column;
```

Description:

Puts the current row and column position of the cursor into the memory locations pointed to by the `row` and `column` parameters, respectively.

Return Values:

None

See Also:

```
set_cursor()
```

Example:

```
main()
{
   int row, col;

   get_cursor(&row, &col);
   printf("Prior to this printf(), the cursor was at row %d, column %d.",row, col);
   getch();
}
```

get_data_source( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_data_source(data_source, data_file)
char *data_source;
char *data_file;
```

Description:

Puts the constant value of the currently selected data source device into the memory location pointed to by the `data_source` parameter. The data source device, which may be defined as either the line or one of the protocol analyzer's discs, indicates from where the event message information for the `read_message` function should be read. The valid constant values are:

data source	value
LINE_RUN	0x2
DISC_RUN	0x3

If `data_source` contains the `DISC_RUN` value, the `data_file` parameter points to a character string representing the path name and file name of the data source. The path name determines which disc has been selected. The protocol analyzers' internal drives are identified by `a:` (the flexible disc drive) and `c:` (the hard disc drive), but any external drives may be used as well. See the *DataCommC User's Guide* for a list of possible disc drive path names.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

get_data_source( )

See Also:

set_data_source()

Example:

```
#include <dlib.include>

main()
{
    char source;
    char filename[81];
    char *source_str[2];

    source_str[0] = "Line";
    source_str[1] = "Disc";

    get_data_source(&source,filename);
    printf("The system processes data from the %s.\n",source_str[source-2]);

    if(source == DISC_RUN)
        printf("The file name is %s.",filename);
    getch();
}
```

get_datacode( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_datacode(datacode)
int *datacode;
```

Description:

Puts a constant value representing the currently selected datacode into the memory location pointed to by the `datacode` parameter. The following `datacode` values are located in the `dlib.include` file.

datacode	value
ASCII8	0x0001
ASCII7	0x0002
EBCDIC	0x0003
HEX8	0x0004
HEX7	0x0005
HEX6	0x0006
HEX5	0x0007
USER_DEF	0x0008

These datacode values can be modified using the `set_protocol` function. The `get_datacode` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

get_datacode( )

See Also:

set_protocol( )

Example:

```
#include <dlib.include>

main()
{
    int datacode;

    get_datacode(&datacode);
    printf("The datacode is %d.",datacode);
    getch();
}
```

get_dir( )

Format:

```
char *get_dir(pathname)
char pathname[82];
```

Description:

Puts a character string representing the path name of the current drive/directory/subdirectory path into the memory location pointed to by the `pathname` parameter.

Return Values:

Returns a pointer to the path name.

See Also:

```
ch_dir(), makedir(), parsedir()
```

Example:

```
main()
{
   char dir[82];

   get_dir(dir);
   printf("The current directory is %s.\n", dir);
   getch();
}
```

get_duplex( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_duplex(duplex)
int *duplex;
```

Description:

Puts the duplex value for the conversation between DTE and DCE devices into the memory location pointed to by the `duplex` parameter. The constant value may be either:

duplex	value
HDX	1
FDX	2

These values are located in the `dlib.include` file.

The duplex values may be modified using the `set_duplex` function. The `get_duplex` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See Also:

`set_duplex()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int duplex;

   set_duplex(FDX);
   get_duplex(&duplex);

   if(duplex != FDX)
   {
      printf("get_duplex() doesn't match the set\n");
   }
   else
         printf("The set_duplex() worked\n");
      getch();
}
```

get_errorcheck( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_errorcheck(errorcheck)
int *errorcheck;
```

Description:

Puts a constant value representing the currently selected errorcheck type into the memory location pointed to by the `errorcheck` parameter. The following errorcheck values are located in the `dlib.include` file.

errorcheck	value
CRC_16	0x1
CRC_CCITT	0x2
NO_ERROR_CHECK	0x3
CRC_12	0x4
LRC	0x5
CRC_6	0x6

These errorcheck values can be modified using `set_protocol()` or the protocol analyzer's Setup Menu. The `get_errorcheck` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

get_errorcheck( )

See Also:

set_protocol()

Example:

```
#include <dlib.include>

main()
{
    int errorcheck;

    get_errorcheck(&errorcheck);
    printf("The errorcheck type is %d.",errorcheck);
    getch();
}
```

get_event_bounds( )

Format:

```
#include <retval.include>
#include <message.include>

int get_event_bounds(num_events, first_event, last_event)

int *num_events;
void **first_event, **last_event;
```

Description:

Puts pointers to the event buffer's first and last events into the memory locations pointed to by the `first_event` and `last_event` parameters, respectively. This function can only be called at the conclusion of a run, that is, after starting and stopping the data link control hardware with the `start_data()` and `stop_data()` functions. The number of complete events is placed in the memory location pointed to by `num_events`. If the buffer is empty, `*num_events` contains zero and the pointers are set to `NULL`.

Return Values:

`SUCCESSFUL (0)`

`ERROR_10 (-10)` The dlc hardware was running.

See Also:

`hold_event(), release_event()`

Example:

```
#include <retval.include>
#include <message.include>
#include <dlib.include>

main()
{
    void *first_event, *last_event;
    MESSAGE *msg;
    int num_events, event_counter = 0;
    int result;

    if( (result = start_data()) != SUCCESSFUL )
    {
       printf("Error starting DLC hardware.\n");
       exit();
    }
    printf("Press a key to stop event collection: ");
    putc('\n');
    sendf("Hello world");    /* make sure the 4954 is configured to transmit */
    getch();
    stop_data();

    get_event_bounds(&num_events, &first_event, &last_event);
    printf("The first event is stored at address %lx hex.\n",first_event);
    printf("The last event is stored at address %lx hex.\n",last_event);
    msg = first_event;
    while (event_counter++ < num_events)
    {
       print_event(msg);     /* call user routine to print the message */
       if(msg++ == BOTTOM_EVENT)
          msg = TOP_EVENT;
    }
    getch();
}
```

get_event_bounds( )

```
print_event(msg) /* user routine to display the event */
MESSAGE *msg;
{
   printf("The message type is %d the subtype is %d.\n",msg->type, msg->subtype);
}
```

get_ignored_leads( )

Format:

```
#include <leads.include>
#include <retval.include>

int get_ignored_leads(lead_id_map)
long *lead_id_map;
```

Description:

Puts the currently selected set of ignored leads for the data link control hardware into the memory location pointed to by the `lead_id_map` parameter. The constants used in `lead_id_map` are defined in the `leads.include` file (see chapter 4 for listings of the include files).

RS232C/MIL188C

lead	value
RTS	0x0001l
CTS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
SQ	0x0040l
DRS	0x0080l
SRS	0x0100l
SCS	0x0200l
SCD	0x0400l

RS449

lead	value
RS	0x0001l
CS	0x0002l
DM	0x0004l
TR	0x0008l
IC	0x0010l
RR	0x0020l
SQ	0x0040l
SI	0x0080l
SRS	0x0100l
SCS	0x0200l
SRR	0x0400l
IS	0x0800l
SF	0x1000l
RL	0x2000l
SS	0x4000l

V.35

lead	value
RS	0x0001l
CS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
LT	0x0040l

get_ignored_leads( )

Return Values:

SUCCESSFUL (0) There is no error return code.

See Also:

set_ignored_leads

Example:

```
#include <leads.include>

main()
{
   long lead_id_map;

   get_ignored_leads(&lead_id_map);

   if(lead_id_map & RTS)
      printf("RTS state changes ignored.\n");
   if(lead_id_map & CTS)
      printf("CTS state changes ignored.\n");
   if(lead_id_map & DSR)
      printf("DSR state changes ignored.\n");
   if(lead_id_map & DTR)
      printf("DTR state changes ignored.\n");
   printf("The lead_id_map is %lx.\n",lead_id_map);
   getch();
}
```

get_lead( )

Format:

```
#include <leads.include>
#include <retval.include>

int get_lead(lead_id, lead_state, message_ptr)
char lead_id, lead_state;
MESSAGE *message_ptr;
```

Description:

Checks to see if the state of the the lead associated with the DATACOMM message at the top of the incoming message queue matches the specified lead_state. The lead can be either ON, OFF, or DONT_CARE. If there is no DATACOMM message in the queue, an error is returned. The constant values available for the lead_id parameter are defined in the leads.include file (see chapter 4 for a complete listing); the names of the constants are shown below:

RS232C/MIL188C

lead_id	value
RTS	0x0001l
CTS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
SQ	0x0040l
DRS	0x0080l
SRS	0x0100l
SCS	0x0200l
SCD	0x0400l

RS449

lead_id	value
RS	0x0001l
CS	0x0002l
DM	0x0004l
TR	0x0008l
IC	0x0010l
RR	0x0020l
SQ	0x0040l
SI	0x0080l
SRS	0x0100l
SCS	0x0200l
SRR	0x0400l
IS	0x0800l
SF	0x1000l
RL	0x2000l
SS	0x4000l

V.35

lead_id	value
RS	0x0001l
CS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
LT	0x0040l

get_lead( )

Return Values:

SUCCESSFUL (0)	The top message (pointed to by message_ptr) is a DATACOMM message with the lead specified by lead_id in the state specified by lead_state.
WARNING_1 (1)	The message at the top of the queue is not a DATACOMM message. The message_ptr parameter is set to point to that message.
WARNING_2 (2)	The lead is not in the specified state. The message_ptr parameter is set to point to the message at the top of the queue.
ERROR_1 (-1)	There wasn't any message in the queue when it was polled. The message_ptr parameter is set to NULL.

See also:

set_lead()

Example:

```
#include <message.include>
#include <leads.include>
#include <dlib.include>
#include <retval.include>

main()
{
   MESSAGE datacomm_message;

   start_data();
   while(1)
   {
      if(is_msg_avail())
      {
         switch(get_lead(RTS,ON,&datacomm_message))
         {
            case SUCCESSFUL:
               printf("RTS has been set ON.\n");
               release_event(datacomm_message.body.event.event_ptr);
               break;
            case WARNING_1:
               printf("The message was not a Datacomm message.\n");
               release_event(datacomm_message.body.event.event_ptr);
               break;
            case WARNING_2:
               release_event(datacomm_message.body.event.event_ptr);
               break;
            case ERROR_1:
               printf("There is no message in the datacomm queue.\n");
               break;
         }
      }
   }
}
```

get_lead_control( )

Format:

```
#include <dlib.include>

int get_lead_control(lead_control)
int *lead_control;
```

Description:

Puts the lead_control value for the conversation between the DTE and DCE devices into the memory location pointed to by the `lead_control` parameter. The constant value is either:

lead_control	value
LEAD_CONTROL_DEFAULT	2
LEAD_CONTROL_USER_DEF	1

These `lead_control` values are located in the `dlib.include` file.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See also:

```
set_lead_control()
```

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int lead_control;

   set_lead_control(LEAD_CONTROL_USER_DEF);
   get_lead_control(&lead_control);

   if(lead_control != LEAD_CONTROL_USER_DEF)
      printf("get_lead_control() doesn't match the set\n");
   else
      printf("set_lead_control() worked\n");
   getch();
}
```

get_mode( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_mode(mode)
int *mode;
```

Description:

Puts a constant value representing the the currently selected synchronization mode into the memory location pointed to by the `mode` parameter. The following mode values are located in the `dlib.include` file.

mode	value
ASYNC_1	0x1
ASYNC_1_5	0x2
ASYNC_2	0x3
SYNC_DCE	0x4
SYNC_NRZI	0x5
SYNC_DTE	0x6

These mode values can be modified using `set_protocol()` or the protocol analyzer's Setup Menu. The `get_mode` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See Also:

`set_protocol()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
    int mode;
    int result;

    if( (result = get_mode(&mode)) != SUCCESSFUL)
    {
        printf("Error #%d returned from get_mode().\n",result);
        getch();
        exit();
    }
    switch (mode)
    {
        case SYNC_DTE:
            printf("The 4954 clock source is the DTE clock.");
            break;
        case SYNC_DCE:
            printf("The 4954 clock source is the DCE clock.");
            break;
        default:
            printf("The value of the synchronization mode is %d.",mode);
    }
    getch();
}
```

get_parity( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_parity(parity)
int *parity;
```

Description:

Puts a constant value representing the the currently selected parity into the memory location pointed to by the `parity` parameter. The following constant values are located in the `dlib.include` file:

parity	value
ODD_PARITY	0x1
EVEN_PARITY	0x2
NO_PARITY	0x3
IGNORE	0x4

These parity values can be modified using `set_protocol()` or the protocol analyzer's Setup Menu. The `get_parity` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See Also:

`set_protocol()`

Example:

```
#include <dlib.include>

main()
{
    int parity;

    get_parity(&parity);
    switch(parity)
    {
        case ODD_PARITY:
            printf("The 4954's parity is odd.");
            break;
        case EVEN_PARITY:
            printf("The 4954's parity is even.");
            break;
        case NO_PARITY:
            printf("The 4954 is not interpreting the parity bit.");
            break;
        case IGNORE:
            printf("The 4954 is ignoring parity.");
            break;
        default:
            printf("The parity type is %d.",parity);
    }
    getch();
}
```

get_protocol( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_protocol(protocol)
int *protocol;
```

Description:

Puts the the value of the currently selected protocol into the memory location pointed to by the `protocol` parameter. The following protocol values are located in the `dlib.include` file.

protocol	value
BSC	0x1
SDLC	0x2
HDLC	0x3
X25	0x4
COPS	0x5
X75	0x7

These protocol values can be modified using the `set_protocol` function or the protocol analyzer's Setup Menu. The `get_protocol` function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)` There is no error return code.

See Also:

`set_protocol()`

Example:

```
#include <dlib.include>

main()
{
    int protocol;

    get_protocol(&protocol);
    switch(protocol)
    {
        case BSC:
            printf("The 4954's protocol is BSC.");
            break;
        case SDLC:
            printf("The 4954's protocol is SDLC.");
            break;
        case HDLC:
            printf("The 4954's protocol is HDLC.");
            break;
        case X25:
            printf("The 4954's protocol is X.25.");
            break;
        case COPS:
            printf("The 4954's protocol is COPS.");
            break;
        case X75:
            printf("The 4954's protocol is X.75.");
            break;
        default:
            printf("The protocol type is %d.",protocol);
    }
    getch();
}
```

get_resyncafter( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_resyncafter(numafter)
int *numafter;
```

Description:

Puts the currently selected resyncafter value into the memory location pointed to by the numafter parameter. When the protocol is COPS and the resync mode is AUTO, the resyncafter value represents the amount of time (in number of characters received) that the protocol analyzer waits after it has received a block check character (bcc) before it drops sync.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The protocol is not COPS.

ERROR_2 (-2) The mode is not AUTO.

See Also:

get_resynchars(), get_resyncmode(), resync(), set_resync(), set_protocol()

Example:

```
#include <dlib.include>

main()
{
    int resync_bytes;

    get_resyncafter(&resync_bytes);
    printf("The 4954 is configured to resync %d bytes after the sync is dropped."
                                                        ,resync_bytes);
    getch();
}
```

get_resynchars( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_resynchars(itext,otext1,otext2,otext3,otext4,otext5,otext6);
char *itext, *otext1, *otext2, *otext3, *otext4, *otext5, *otext6;
```

Description:

Puts the values of the currently selected resynchars (intext and outoftext) into the memory locations pointed to by the `itext` and `otext1-otext6` parameters. When the protocol is `COPS` and resync mode is `AUTO`, the `itext` and `otext1-otext6` values represent characters that trigger the protocol analyzer's drop sync operation.

The `itext` parameter is an 8-bit value that tells the data link control hardware what character to drop sync on while in the transparent text mode. See `set_transtext()` for more information on the transparent text mode.

The `otext` parameters (`otext1 - otext6`) form an array of six 8-bit values defining characters that the data link control hardware drops sync on while outside of the transparent text mode.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` The protocol is not `COPS`.

`ERROR_2 (-2)` The mode is not `AUTO`.

See Also:

`get_resyncafter(), get_resyncmode(), resync(), set_resync(), set_protocol()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   char itext, otext1, otext2, otext3, otext4, otext5, otext6;
   int result;

   if((result =
      get_resynchars(&itext,&otext1,&otext2,&otext3,&otext4,&otext5,&otext6))
      != SUCCESSFUL )
   {
      printf("Error #%d returned from get_resynchars().\n",result);
      exit();
   }
   printf("The 4954 will start to resync after '%c' in the frame or\n",itext);
   printf("one of the following values outside the frame:\n");
   printf("     '%c',\n     '%c',\n     '%c',\n     '%c',\n     '%c',\n     '%c'\n",
        otext1, otext2, otext3, otext4, otext5, otext6);
   getch();
}
```

get_resyncmode( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_resyncmode(rmode)
int *rmode;
```

Description:

Puts the currently selected resync mode value into the memory location pointed to by the rmode parameter. This function should only be used when a character-oriented protocol (COPS in set_protocol( ) or **Char Asyn/Syn** in the Setup Menu) is selected and the mode is synchronous (SYNC_DCE or SYNC_DTE in set_protocol( ) or **Sync** in the Setup Menu). The following resync mode values are located in the dlib.include file:

rmode	value
MANUAL	0x1
AUTO	0x2

The resync mode values can be modified using set_resync( ) or the protocol analyzer's Setup Menu. The get_resyncmode function may be called while the data link control hardware is running.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The protocol is not COPS.

ERROR_2 (-2) The mode is not SYNC_DCE or SYNC_DTE.

get_resyncmode( )

See Also:

get_resyncafter(), get_resynchars(), resync(), set_resync(), set_protocol()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   unsigned int rmode;
   int result;

   if((result = get_resyncmode(&rmode)) != SUCCESSFUL)
   {
      printf("Error #%d returned from get_resyncmode().\n", result);
      getch();
      exit();
   }

switch(rmode)
   {
      case MANUAL:
         printf("The 4954 is set to resync manually.");
         break;
      case AUTO:
         printf("The 4954 is set to resync automatically.");
         break;
      default:
         printf("The value of the resynchronization mode is %d.",rmode);
   }
   getch();
}
```

get_startbcc( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_startbcc(startbcc1,startbcc2)
char *startbcc1, *startbcc2;
```

Description:

Puts the currently selected startbcc characters into the memory locations pointed to by the `startbcc1` and `startbcc2` parameters. These parameters are used to start the block check character error-check calculations.

This function should only be used when a character-oriented protocol (`COPS` in `set_protocol()` or **Char Asyn/Syn** in the Setup Menu) is selected and error checking is active (anything but `NO_ERROR_CHECK` in `set_protocol()` or **None** in the Setup Menu).

The startbcc values may be modified using `set_bcc()` or the protocol analyzer's Setup Menu. The get_startbcc function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` The protocol is not `COPS`.

`ERROR_2 (-2)` The errorcheck is `NO_ERROR_CHECK`.

See Also:

`set_bcc(), get_stopbcc()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   char startbcc1, startbcc2;
   int result;

   if((result = get_startbcc(&startbcc1, &startbcc2)) != SUCCESSFUL)
   {
      printf("Error #%d returned from get_startbcc().\n",result);
      exit();
   }
   printf("The BCC starts after either '%c' and '%c'.\n",startbcc1, startbcc2);
   getch();
}
```

get_stop_states( )

Format:

```
#include <dlib.include>          /* for ON, OFF */
#include <leads.include>      /* for ACTIVE, NOT_DRIVEN, and all lead_id defs */
#include <retval.include>

int get_stop_states(lead_id, stop_state)
int lead_id, *stop_state;
```

Description:

Puts the current stop state for the specified lead into the memory location pointed to by the `stop_state` parameter. The lead id values and some of the stop state values are defined in the `leads.include` file, and the remaining stop state values can be found in `dlib.include`.

stop_state	value
OFF	0x0
ON	0x1
NOT_DRIVEN	0x2
MARKING	0x3
ACTIVE	0x4

The stop state values may be modified using the `set_stop_states` function.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` The `lead_id` parameter was invalid.

`ERROR_2 (-2)` The value returned for the requested `stop_state` was invalid

get_stop_states( )

See Also:

set_stop_states()

Example:

```
#include <dlib.include>
#include <leads.include>
#include <retval.include>

main()
{
   int result;
   int stop_state;

   if((result = get_stop_states(RTS, &stop_state)) != SUCCESSFUL)
   {
      printf("Error #%d returned from get_stop_states().\n",result);
      exit();
   }

   switch(stop_state)
   {
      case(ON):
         printf("The RTS will be on after stop_data() is called.");
         break;
      case(OFF):
         printf("The RTS will be off after stop_data() is called.");
         break;
      case(NOT_DRIVEN):
         printf("The RTS will be not driven after stop_data() is called.");
         break;
   }
   getch();
}
```

get_stopbcc( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_stopbcc(stopbcc1, stopbcc2)
char *stopbcc1, *stopbcc2;
```

Description:

Puts the currently selected stopbcc characters into the memory locations pointed to by the `stopbcc1` and `stopbcc2` parameters. These parameters are used to stop the block check character error-check calculations.

This function should only be used when a character-oriented protocol (`COPS` in `set_protocol()` or **Char Asyn/Syn** in the Setup Menu) is selected and error checking is active (anything but `NO_ERROR_CHECK` in `set_protocol()` or **None** in the Setup Menu).

The stopbcc values may be modified using `set_bcc()` or the protocol analyzer's Setup Menu. The get_stopbcc function may be called while the data link control hardware is running.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` The protocol is not `COPS`.

`ERROR_2 (-2)` The errorcheck is `NO_ERROR_CHECK`.

See Also:

`set_bcc(), get_startbcc()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   char stopbcc1, stopbcc2;
   int result;

   if((result = get_stopbcc(&stopbcc1, &stopbcc2)) != SUCCESSFUL)
   {
      printf("Error #%d returned from get_stopbcc().\n",result);
      exit();
   }

   printf("The stop BCC characters are %x and %x.\n",stopbcc1, stopbcc2);
   getch();
}
```

get_synchars( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_synchars(mode, synchar1, synchar2)
int *mode;
char *synchar1, *synchar2;
```

Description:

Puts the currently selected sync chars into the memory locations pointed to by the synchar1 and synchar2 parameters. The value of the current synchronization mode is placed in the memory location pointed to by the mode parameter. This function should only be used when a character-oriented protocol (COPS in set_protocol() or Char Asyn/Syn in the Setup Menu) is selected and the mode is synchronous (SYNC_DCE or SYNC_DTE in set_protocol() or Sync in the Setup Menu).

The synchar values may be modified using set_sync() or the protocol analyzer's Setup Menu. The get_synchars function may be called while the data link control hardware is running.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The protocol is not COPS.

ERROR_2 (-2) The mode is not synchronous (must be either SYNC_DCE or SYNC_DTE).

See Also:

set_sync()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
    int mode;
    char synchar1, synchar2;
    int result;

    if((result = get_synchars(&mode,&synchar1,&synchar2)) != SUCCESSFUL)
    {
        printf("Error #%d returned from get_synchars().\n",result);
        exit();
    }
    printf("The sync characters are %x(hex) and %x(hex), for mode %d.\n"
        ,synchar1, synchar2,mode);
    getch();
}
```

get_time( )

Format:

```
#include <time.include>
#include <retval.include>

void get_time()
struct tm *time;
```

Description:

Captures the current system time from the real-time-clock and writes it to the data structure pointed to by the `time` parameter. The `tm` data structure is defined in the `time.include` file (see chapter 4, "Include Files", for more information), and is outlined below.

struct tm

int	tm_msecs	0-900 milliseconds after the second (100 msec intervals)
int	tm_sec	0-59 seconds after the minute
int	tm_min	0-59 minutes after the hour
int	tm_hour	0-23 hours after midnight
int	tm_day	1-31 days of the month
int	tm_mon	1-12 months of the year
int	tm_year	0-99 years after turn of century

Return Values:

None

get_time( )

See Also:

set_time()

Example:

```
#include <time.include>

main()
{
    struct tm time;

    get_time(&time);
    printf("The date is %d/%d/%d and the time is %d:%d:%d.%d.",
        time.tm_mon,
        time.tm_day,
        time.tm_year,
        time.tm_hour,
        time.tm_min,
        time.tm_sec,
        time.tm_msecs );
    getch();
}
```

get_transtext( )

Format:

```
#include <dlib.include>
#include <retval.include>

int get_transtext(mode,ttextchar)
int *trans_mode;
char *ttextchar;
```

Description:

Puts the currently selected transparent text character into the memory location pointed to by the ttextchar parameter. The trans_mode parameter gets the current value of the transparent text mode, which may be either ON (1) or OFF (0).

The transparent text character, ttextchar, is an 8-bit value that defines the boundaries of the transparent text mode. It signals both the beginning and the end of the transparent text mode. While transparent text mode is active (trans_mode is ON), all control characters are treated as data, that is, their defined control functions are not executed.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The protocol is not COPS.

See Also:

set_transtext()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int mode;
   char transtext;
   int result;

   if((result = get_transtext(&mode,&transtext)) != SUCCESSFUL)
   {
      printf("Error #%d returned from get_transtext().\n",result);
      exit();
   }
   printf("The transparent text character is '%c', mode = %d\n",transtext,mode);
   getch();
}
```

getc( ), getchar( )

Format:

```
#include <stdio.include>

int getc(file)
FILE *file;

int getchar()
```

Description:

Reads a character from a file or input stream. The `getc` function looks for input from whatever file or stream was specified by `file`, while `getchar()`, a macro found in the `stdio.include` file, takes its input only from the standard input stream.

As a character is read, `getc()` and `getchar()` lock the keyboard so that no other process can gain access to it. In this way, a process is prevented from collecting information that was not intended for it. The keyboard can also be locked by calling `keyboard_lock()`.

A process that attempts to read from the keyboard when it has already been locked is blocked until the process that locked the keyboard releases it by calling `keyboard_unlock()`, or by terminating itself.

The keyboard functions that lock the keyboard are: `fscanf()`, `fgets()`, `getc()`, `getch()`, `getchar()`, `gets()`, `is_key_avail()`, `scanf()`, and `ungetch()`.

Return Values:

Returns the character read from the file or input stream.

See Also:

`getch()`

Example:

```
#include <stdio.include>

main()
{
    int ch;

    printf("Press a key, then press RETURN:\n");
    ch = getc(stdin);
    printf("The value of the key pressed is: %d.\n",ch);
    getch();
}
```

getch( )

Format:

```
#include <conio.include>

int getch()
```

Description:

Reads an unbuffered character from the keyboard without echoing the key to the screen. As it reads a character, `getch()` locks the keyboard so that no other process can gain access to it. In this way, a process is prevented from collecting information that was not intended for it. The keyboard can also be locked by calling `keyboard_lock()`.

A process that attempts to read from the keyboard when it has already been locked is blocked until the process that locked the keyboard releases it by calling `keyboard_unlock()`, or by terminating itself.

The keyboard functions that lock the keyboard are: `fscanf()`, `fgets()`, `getc()`, `getch()`, `getchar()`, `gets()`, `is_key_avail()`, `scanf()`, and `ungetch()`.

Return Values:

Returns the character read from the keyboard.

See Also:

`getc()`, `ungetch()`

Example:

```
#include <conio.include>

main()
{
   int ch;

   while((ch = getch()) != CLEAR_DISPLAY) /* CLEAR_DISPLAY is in conio.include */
      printf("The value of the key pressed is: %d.\n",ch);
}
```

gets( )

Format:

```
char *gets(buffer)
char *buffer;
```

Description:

Reads a character string from the standard input stream, stdin, and places it in the buffer pointed to by `buffer`. The input string consists of all characters received up to and including the first newline character (`'\n'`). The `gets` function replaces the newline character with a null character (`'\0'`) which is placed in the buffer. `fgets()`, on the other hand, retains the newline character and stores it in `buffer`.

As a character string is read, `gets()` locks the keyboard so that no other process can gain access to it. In this way, a process is prevented from collecting information that was not intended for it. The keyboard can also be locked by calling `keyboard_lock()`.

A process that attempts to read from the keyboard when it has already been locked is blocked until the process that locked the keyboard releases it by calling `keyboard_unlock()`, or by terminating itself.

The keyboard functions that lock the keyboard are: `fscanf()`, `fgets()`, `getc()`, `getch()`, `getchar()`, `gets()`, `is_key_avail()`, `scanf()`, and `ungetch()`.

Return Values:

Returns a pointer to the string read from the keyboard, or a `NULL` pointer if an error occurred.

See Also:

`fgets()`, `puts()`, `getch()`

Example:

```
main()
{
   char instr[70];

   gets(instr);
   printf("The string read from stdin is %s.", instr);
   getch();
}
```

hold_event( )

Format:

```
#include <message.include>
#include <retval.include>

int hold_event(event)
void *event;
```

Description:

Retains access to the event specified by the event parameter even after sending it to another process (see send_message( )). This is accomplished by incrementing the held_count parameter of the specified event.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) Event not found

See Also:

read_message( ), send_message( ), release_event( )

Example:

```
#include <dlib.include>
#include <retval.include>
#include <message.include>

main()
{
    MESSAGE message;
    int message_counter = 0;

    start_data();

    while(1)
    {
        if(read_message(&message,WAIT,WAIT_FOREVER))
            printf("Error reading queue.\n");
        else
        {
            message_counter += 1;
            printf("message #%d read.\n",message_counter);
            if ( message_counter == 10 )
                hold_event(message.body.event.event_ptr);
            else
                release_event(message.body.event.event.ptr);
        }
    }
}
```

init_rs232( )

Format:

```
void init_rs232()
```

Description:

Initializes the protocol analyzer's serial port. This function must be called prior to calling `print_char()` or any of the other RS232 functions.

Return Values:

None

See Also:

```
rs232_write_ready(), rs232_read_ready(), read_rs232(), write_rs232(), print_char()
```

Example:

```
main()
{
   init_rs232();

   printf("the rs232 port is now initialized!\n");
   getch();
}
```

init_trigger( )

Format:

```
int init_trigger(message_type, number_of_subtypes, subtype_args)
unsigned message_type;
unsigned number_of_subtypes;
unsigned subtype_args;
```

Description:

Initializes the system `trigger_on_message` function to trigger on a message of type `message_type` with a subtype that is one of the subtypes in the argument list `subtype_args`. The `number_of_subtypes` parameter represents the number of subtypes that follow in the `subtype_args` list. Individual subtype values can range from 1 to 31.

If the `number_of_subtypes` parameter is zero, `init_trigger()` clears all subtype triggers that were set for the specified `message_type`. If the `number_of_subtypes` parameter is greater than 31, the `init_trigger` function sets all subtypes (of a certain `message_type`) to triggers. In these two cases, `init_trigger()` ignores any subtypes listed in `subtype_args`.

Return Values:

Returns the number of new subtype triggers set by the call to `init_trigger()`.

See Also:

```
trigger_on_message()
```

init_trigger( )

Example:

```
#include <message.include>
#define The_sun_sets_in_the_west 1

main()
{
    MESSAGE message;
    int number_set;

    number_set = init_trigger(DATACOMM, 2 , DTE_FRAME, END_OF_DISC_RUN);
    printf("There are %d subtypes set by init_trigger for DATACOMM events.\n",
                                                                    number_set);
    number_set = init_trigger(TIMER, 32);
    printf("There are %d subtypes set by init_trigger for timer messages.\n",number_set);
    set_timer(10L);  /* timeout after 1 second */
    start_data(); /* start receiving DATACOMM events */

    while(The_sun_sets_in_the_west)
        switch(trigger_on_message(&message))
        {
            case DATACOMM:
                printf("Process DATACOMM messages here.\n");
                break;
            case TIMER:
                printf("Process TIMER messages here.\n");
                break;
            default: /* must be an error subtype */
                printf("Process ERROR messages here.\n");
        }
    getch();
}
```

is_key_avail( )

Format:

```
int is_key_avail();
```

Description:

Polls the keyboard buffer for available keys. This function locks the keyboard when called (if the keyboard isn't already locked), so that unless `keyboard_unlock()` is called directly, or the process terminates, no other process can gain access to the keyboard after `is_key_available()` has been called. If another process has gained control of the keyboard, however, is_key_available does not wait for the keyboard to be released; it returns without blocking.

Return Values:

Returns the number of keys pressed.

See Also:

```
getch(), keyboard_lock(), keyboard_unlock()
```

Example:

```
main()
{
   int ch;

   while( !is_key_avail() )
      printf("Waiting for a keystroke.\n");
   ch = getch();
   printf("The hex value of the key pressed is %x.\n",ch);
   getch();
}
```

is_msg_avail( )

Format:

```
int is_msg_avail();
```

Description:

Polls the process message queue for an available message.

Return Values:

Returns the number of messages available in the queue.

See Also:

```
read_message(), send_message()
```

Example:

```
#include <dlib.include>
#include <retval.include>
#include <message.include>

main()
{
   MESSAGE message;
   int message_counter = 0;

   start_data();

   while(!is_msg_avail())  /* wait until a msg is available */
      printf("Waiting for a message.\n");

   if(read_message(&message,WAIT,WAIT_FOREVER))
      printf("Error reading queue.\n");
   else
   {
      printf("message read type/subtype fields are %d/%d .\n",
         message.type, message.subtype);
      release_event(message.body.event.event_ptr);
   }
   getch();
}
```

isalnum( )

Format:

```
#include <ctype.include>

int isalnum(ch)
int ch;
```

Description:

Tests for an alphanumeric character ('A' through 'Z', 'a' through 'z', or '0' through '9'). The `isalnum` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is an alphanumeric character, zero if it is not.

See Also:

`isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace,(), isupper(), isxdigit(), toascii(), tolower(), toupper()`

Example:

```
#include <ctype.include>
#include <stdio.include>

main()
{
    int ch;

    printf("Press a key, then press RETURN. Press any key to exit.");
    while((ch = getc(stdin)) != '\n')
    {
```

```
        puts("*************************************************");
        if(isalnum(ch))
           printf("The character '%c' is alphanumeric.\n",ch);
        if(isalpha(ch))
           printf("The character '%c' is alphabetic.\n",ch);
        if(isascii(ch))
           printf("The character '%c' is an ASCII character.\n",ch);
        if(iscntrl(ch))
           printf("The character '%c' is a control character.\n",ch);
        if(isdigit(ch))
           printf("The character '%c' is a numeric digit.\n",ch);
        if(isgraph(ch))
           printf("The character '%c' is a graphics character.\n",ch);
        if(islower(ch))
           printf("The character '%c' is in lower case.\n",ch);
        if(isprint(ch))
           printf("The character '%c' is a printable character.\n",ch);
        if(ispunct(ch))
           printf("The character '%c' is a punctuation character.\n",ch);
        if(isspace(ch))
           printf("The character '%c' is a white space character.\n",ch);
        if(isupper(ch))
           printf("The character '%c' is in upper case.\n",ch);
        if(isxdigit(ch))
           printf("The character '%c' is a hex digit.\n",ch);
    }
    getch();
}
```

isalpha( )

Format:

```
#include <ctype.include>

int isalpha(ch)
int ch;
```

Description:

Tests for an alphabetic character ('A' through 'Z', or 'a' through 'z'). The `isalpha` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is an alphabetic character, zero if it is not.

See Also:

`isalnum()`, `isascii()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace,()`, `isupper()`, `isxdigit()`, `toascii()`, `tolower()`, `toupper()`

Example:

See the example in `isalnum()`.

isascii( )

Format:

```
#include <ctype.include>

int isascii(ch)
int ch;
```

Description:

Tests for an ASCII character (0x00 through 0x7F). The isascii macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if ch is an ASCII character, zero if it is not.

See Also:

isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace,(), isupper(), isxdigit(), toascii(), tolower(), toupper()

Example:

See the example in isalnum().

iscntrl( )

Format:

```
#include <ctype.include>

int iscntrl(ch)
int ch;
```

Description:

Tests for a control character (`0x00` through `0x1F`, as well as `0x7F`). The `iscntrl` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is a control character, zero if it is not.

See Also:

`isalnum()`, `isalpha()`, `isascii()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace,()`, `isupper()`, `isxdigit()`, `toascii()`, `tolower()`, `toupper()`

Example:

See the example in `isalnum()`.

isdigit( )

Format:

```
#include <ctype.include>

int isdigit(ch)
int ch;
```

Description:

Tests for a numeric digit ('0' through '9'). The `isdigit` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is a numeric digit, zero if it is not.

See Also:

`isalnum()`, `isalpha()`, `isascii()`, `iscntrl()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace,()`, `isupper()`, `isxdigit()`, `toascii()`, `tolower()`, `toupper()`

Example:

See the example in `isalnum()`.

isgraph( )

Format:

```
#include <ctype.include>

int isgraph(ch)
int ch;
```

Description:

Tests for a printable character excluding the space character (0x21 through 0x7E). The isgraph macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if ch is a graphic character, zero if it is not.

See Also:

isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), islower(), isprint(), ispunct(), isspace,(), isupper(), isxdigit(), toascii(), tolower(), toupper()

Example:

See the example in isalnum().

islower( )

Format:

```
#include <ctype.include>

int islower(ch)
int ch;
```

Description:

Tests for a lowercase character ('a' through 'z'). The `islower` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is a lowercase character, zero if it is not.

See Also:

`isalnum()`, `isalpha()`, `isascii()`, `iscntrl()`, `isdigit()`, `isgraph()`, `isprint()`, `ispunct()`, `isspace,()`, `isupper()`, `isxdigit()`, `toascii()`, `tolower()`, `toupper()`

Example:

See the example in `isalnum()`.

isprint( )

Format:

```
#include <ctype.include>

int isprint(ch)
int ch;
```

Description:

Tests for a printable character (0x20 through 0x7E). The isprint macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if ch is a printable character, zero if it is not.

See Also:

isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), ispunct(), isspace,(), isupper(), isxdigit(), toascii(), tolower(), toupper()

Example:

See the example in isalnum().

ispunct( )

Format:

```
#include <ctype.include>

int ispunct(ch)
int ch;
```

Description:

Tests for a punctuation character (0x21 through 0x2F, 0x3A through 0x40, 0x5B through 0x60, or 0x7B through 0x7E). The ispunct macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if ch is a punctuation character, zero if it is not.

See Also:

isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), isspace,(), isupper(), isxdigit(), toascii(), tolower(), toupper()

Example:

See the example in isalnum().

isspace( )

Format:

```
#include <ctype.include>

int isspace(ch)
int ch;
```

Description:

Tests for a white space character (0x09 through 0x0D, or 0x20). The isspace macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if ch is a white space character, zero if it is not.

See Also:

isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isupper(), isxdigit(), toascii(), tolower(), toupper()

Example:

See the example in isalnum().

isupper( )

Format:

```
#include <ctype.include>

int isupper(ch)
int ch;
```

Description:

Tests for a uppercase character ('A' through 'Z'). The `isupper` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is a uppercase character, zero if it is not.

See Also:

`isalnum()`, `isalpha()`, `isascii()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace,()`, `isxdigit()`, `toascii()`, `tolower()`, `toupper()`

Example:

See the example in `isalnum()`.

isxdigit( )

Format:

```
#include <ctype.include>

int isxdigit(ch)
int ch;
```

Description:

Tests for a hexadecimal digit ('A' through 'F', 'a' through 'f', or '0' through '9'). The `isxdigit` macro expects 8 bit values as input.

Return Values:

Returns an integer greater than zero if `ch` is a hexadecimal digit, zero if it is not.

See Also:

`isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace,(), isupper(), toascii(), tolower(), toupper()`

Example:

See the example in `isalnum()`.

itoa( )

Format:

```
#include <stdlib.include>

char *itoa (num,str,radix)

int num;
char *str;
int radix;
```

Description:

Converts the value of `num` to a null-terminated character string of digits and stores the result in the memory location pointed to by the `str` parameter. The `radix` parameter specifies the base of the argument to be converted.

Return Values:

Returns address of `str`.

See Also:

`atoi(), ftoa() ltoa()`

itoa( )

Example:

```
#include <stdlib.include>

main()
{
    int i = 20;
    char str[10];

    itoa(i,str,16);
    printf("The base 16 representation of %d is %s.\n",i,str);
    getch();
}
```

keyboard_lock( )

Format:

```
void keyboard_lock()
```

Description:

Locks the keyboard for exclusive use by the current process. If the keyboard has been locked by another process prior to the function call, the current calling process is blocked until it can lock the keyboard.

Several general I/O functions, `is_key_avail()`, `getch()`, `gets()`, `getc()`, `getchar()`, `scanf()` and `ungetch()` call this function indirectly and block other processes from keyboard access. `fgets()`, `fgetc()`, `fscanf()` and `fread()` call `keyboard_lock()` when the file pointer is `stdin`; `read()` locks the keyboard when its file descriptor is zero. In each of these cases, the keyboard is locked until `keyboard_unlock()` is called or until the calling process terminates.

Return Values:

None

See Also:

`keyboard_unlock()`

keyboard_lock( )

Example:

```
main()
{
    keyboard_lock();
    printf("The keyboard is now locked for this process - press a key.\n");
    getch();
    keyboard_unlock();
    printf("The keyboard is not locked for this process.\n");
    wait(30l);
}
```

keyboard_unlock( )

Format:

```
int keyboard_unlock()
```

Description:

If called by the process that locked the keyboard, allows another process to take control of the keyboard. If another process had called `keyboard_lock()`, and was blocked, it is unblocked.

Return Values:

Returns a `1` if the process successfully unlocked the keyboard.

See Also:

`keyboard_lock()`

Example:

See the example in `keyboard_lock()`.

log( )

Format:

```
#include <math.include>

double log(x)
double x;
```

Description:

Returns the natural logarithm of x.

Return Values:

Returns the result of the log function.

See Also:

```
log10()
```

Example:

```
#include <math.include>

main()
{
   double x = 10,y;

   y = log(x);
   printf("The log of %f is %f.",x,y);
   getch();
}
```

log10( )

Format:

```
#include <math.include>

double log10(x)
double x;
```

Description:

Returns the base 10 log of x.

Return Values:

Returns the result of the log10 function.

See Also:

```
log()
```

Example:

```
#include <math.include>

main()
{
   double x = 10,y;

   y = log10(x);
   printf("The log (base 10) of %f is %f.",x,y);
   getch();
}
```

longjmp( )

Format:

```
#include <setjmp.include>

void longjmp(env, val)
jmp_buf env;
int val;
```

Description:

Restores the stack environment values (env) previously saved by a call to `setjmp()`, and causes program execution to continue as if the call to `setjmp()` was just terminating with `val` as its return code. The `val` parameter must be a nonzero number. The `longjmp` and `setjmp` functions are useful for dealing with errors encountered by the low-level functions of a program.

The `longjmp` function must not be called unless `env` has already been initialized by a call to `setjmp()`. It also must not be called if the function that called `setjmp()` has returned.

Return Values:

None

See Also:

`setjmp()`

Example:

```
#include <setjmp.include>

main()
{
   jmp_buf buff;

   printf("We are about to call setjmp.\n");
   printf("Please hit any character to call it.\n");
   getch();
   if(setjmp(buff))
   {
      printf("We are in the if clause.\n");
      printf("We should have had an error and will return.\n");
   }
   else
   {
      printf("We are executing the else clause.\n");
      printf("We are now pretending to have an error.\n");
      printf("Hit any key to have a fake error and call longjmp.\n");
      getch();
      longjmp(buff,-1);
      printf("We have just called longjmp.  This should never get printed.\n");
   }
   printf("Please hit any key to return.\n");
   getch();
}
```

lseek( )

Format:

```
#include <stdio.include>

long lseek(fdesc, offset, origin)
int fdesc;
long offset;
int origin;
```

Description:

Moves the file pointer `fdesc` to a location in the file `offset` bytes from the `origin`. The `fdesc` parameter must be the file descripter returned by `open()`. Constant values for the `origin` parameter are defined in the `stdio.include` file, and are described below:

origin	Description
SEEK_SET	The file pointer indicates the beginning of the file.
SEEK_CUR	The file pointer indicates the current position in the file.
SEEK_END	The file pointer indicates the end of the file.

Return Values:

Returns zero if the function was successful, or -1 if an error occurred during the operation.

See Also:

`fseek(), open(), close()`

Example:

```
#include <stdio.include>
#include <fcntl.include>

main()
{
    int fdesc;
    long seek_result, write_result;
    char instr[20];

    if((fdesc = open("file1.text",O_WRONLY|O_CREAT)) == -1)
       printf("Can't create file1.text.");
    else
    {
       if(write_result = write(fdesc,"hello world\n",12) != 12)
          printf("Write error detected.");
       else
       {
          close(fdesc);
          if((fdesc = open("file1.text",O_RDONLY)) == -1)
             printf("Can't open file1.text.");
          else if ((seek_result = lseek(fdesc,6L,SEEK_SET)) != 6L)
             printf("Seek error detected.");
          else
          {
             read(fdesc,instr,3);
             close(fdesc);
             instr[3] = 0;
             printf("the string read after lseek is %s.",instr);
          }
       }
    }
    getch();
}
```

ltoa( )

Format:

```
#include <stdlib.include>

char *ltoa(num,str,radix)

long num;
char *str;
int radix;
```

Description:

Converts the digits in `num` to a null terminated character string and stores the result in the memory space pointed to by the `str` parameter. The `radix` parameter specifies the base of the argument to be converted. Returns the address of `str`.

Return Values:

Returns the address of `str`. There is no error return value.

See Also:

`atol(), itoa()`

Example:

```
#include <stdlib.include>

main()
{
   long i = 20;
   char str[10];

   ltoa(i,str,16);
   printf("The base 16 representation of %ld is %s.\n",i,str);
   getch();
}
```

make_file( )

Format:

```
make_file(pathname, size)
char *pathname;
unsigned long size;
```

Description:

Creates a file, using the value of the `size` parameter as the length of the new file. The `pathname` parameter must contain the full device/directory/filename path.

Return Values:

0	The file was created successfully.
1	An error occurred during the operation.

See also:

```
unlink()
```

Example:

```
main()
{
   if(make_file("c:makefile.text",2048L))
      printf("Error creating file c:makefile.text.\n");
   else
      printf("Created file:  c:makefile.text; size:  2048 bytes");
   getch();
}
```

makedir( )

Format:

```
char *makedir(rtnpath, dev, path, filename, filetype)
char rtnpath[82];
char dev;
char *path;
char *filename;
char *filetype;
```

Description:

Creates a string representing a directory path by combining the `dev`, `path`, `filename` and `filetype` parameters. The function expects to have the device supplied as a single character (for example, `'c'`), and the path, file name and file extension as a character arrays. The makedir function puts the directory path string into the memory location indicated by the `rtnpath` parameter.

The `makedir` function is useful for assembling path strings that may be used in `ch_dir()`, `fopen()`, `open()`, or other file-oriented functions. If the `dev` or `path` parameters are replaced with `NULL` characters, values from the current directory are substituted, so that only the file name and extension are changed. The current directory may be changed with `ch_dir()`.

Return Values:

Returns a pointer to the newly created directory path string.

See Also:

`get_dir()`, `parsedir()`

makedir( )

Example:

```
main()
{
    char rtnpath[82];

    makedir(rtnpath,'c',"c:/","makefile","text");
    printf("The string returned from makedir is %s.\n",rtnpath);
    getch();
}
```

malloc( )

Format:

```
#include <stdlib.include>

char *malloc(size)
unsigned int size;
```

Description:

Allocates `size` bytes of memory. The allocated block (`size`) cannot exceed 32K bytes.

Return Values:

Returns a pointer to the allocated memory or `NULL` if the memory cannot be allocated. When memory allocated in this way is no longer being used, it should be returned to the system by calling `free()`.

See Also:

```
calloc(), free()
```

malloc( )

Example:

```
#include <stdlib.include>
#include <stdio.include>

main()
{

   char *str, *new_str;

   str="hello world";
   if((new_str = malloc(strlen(str) + 1))!= NULL)
   {
      strcpy(new_str, str);
      printf("The new_str is %s.", new_str);
   }
   getch();
}
```

open( )

Format:

```
#include <stdio.include>
#include <fcntl.include>

int open(filename, access_mode)
char *filename;
int access_mode;
```

Description:

Opens the file whose name is a string pointed to by the `filename` parameter, using the `access_mode` parameter to determine the type of file access allowed. Values for the `access_mode` parameter are located in the `fcntl.include` file. These values, represented here by their associated names, may be joined together with the aritmetic OR operator (|), but since there is no default, one of the values given must be either `O_RDONLY`, `O_WRONLY`, or `O_RDWR`. These three values are also mutually exclusive; only one of them may be selected during a file access.

access_mode	Description
O_RDONLY	The file is opened in read-only mode.
O_WRONLY	The file is opened in write-only mode.
O_RDWR	The file is opened in read/write mode.

open( )

access_mode	Description
O_CREAT	Creates a new file. This has no effect if the file specified by filename already exists.
O_EXCL	Causes O_CREAT to return an error message if the specified path already exists. Use only with O_CREAT.
O_TRUNC	Truncates an existing file to zero length. The file must be opened in write-only or read/write mode. File contents are destroyed.
O_APPEND	Moves the file pointer to the end of the file before every write operation.

Note The O_TRUNC access_mode option destroys all information in an existing file; it should be used with extreme caution.

Regardless of the access mode used, an existing file is not allowed to grow beyond the physical size allocated to it when it was created. If open() is used to create a file, the file size cannot be specified, but if the file is created with make_file(), a file size can be specified. Any file, therefore, that might be added on to in the future should be created with make_file().

Return Values:

Returns a file descriptor, an integer that represents the file, when successful, or -1 if the file could not be opened.

See Also:

close(), fopen(), make_file()

Example:

```
#include <stdio.include>
#include <fcntl.include>

main()
{
    int fdesc;
    long seek_result, write_result;
    char instr[20];

    if((fdesc = open("file1.text",O_WRONLY)) == -1)
       printf("Can't open file1.text.");
    else
    {
       if((write_result = write(fdesc,"hello world\n",12)) != 12)
          printf("Write error detected.");
       close(fdesc);
    }
getch();
}
```

parsedir( )

Format:

```
void parsedir(name, dev, path, filename, filetype)
char *name;
char dev[3];
char path[56];
char filename[11];
char filetype[14];
```

Description:

Breaks the path supplied by `name` into its components. The function reads in the character array, `name` (up to 82 characters long) and puts the individual pieces into `dev` (for example, `'a:'` or `'c:'`), `path`, `filename` and `filetype` (for example, `.text`, `.backup`). All of these parameters are represented by character arrays.

If the `name` parameter does not include a device or directory path, the associated parameter or parameters (`dev`, `path`) are supplied with values taken from the current path.

Return Values:

None

See Also:

`makedir()`, `get_dir()`

Example:

```
main()
{
   char name[82];
   char dev[3];
   char path[56];
   char filename[11];
   char type[14];

   parsedir("c:C/User/Csource/Prog1.csource", dev,path,filename,type);
   printf("The components of the full path follow:\n");
   printf("  The device is: %s.\n",dev);
   printf("    The path is: %s.\n",path);
   printf("The filename is: %s.\n",filename);
   printf("    The type is: %s.\n",type);
   getch();
}
```

pow( )

Format:

```
#include <math.include>

double pow(x, y)
double x, y;
```

Description:

Raises x to the power of y.

Return Values:

Returns the result of the operation.

See Also:

```
exp()
```

Example:

```
#include <math.include>

main()
{
   double x = 10,y = 3,powx;

   powx = pow(x,y);
   printf("The value of %f raised to the power of %f is %f.",x,y,powx);
   getch();
}
```

print_char( )

Format:

```
void print_char(chr)
char chr;
```

Description:

Prints the given character to the RS232 port. This function should only be used after `init_rs232()` has been called.

Return Values:

None

See Also:

`init_rs232()`, `rs232_write_ready()`, `write_rs232()`

Example:

```
#include <stdio.include>

main()
{
    int ch = 0;

    init_rs232();
    printf("Type text to send to printer and press RETURN - enter \"*\" to exit\n\n");
    while((ch = getc(stdin)) != '*')
        print_char(ch);
}
```

printf( )

Format:

```
void printf(format,args)
char *format;
char *args;
```

Description:

Outputs arguments in `args` to the standard output device, `stdout`, according to `format`.

The character string pointed at by `format` directs the output operation, and contains two types of information: ordinary alphanumeric characters, which are output unchanged; and conversion specifications, each of which causes the conversion and output of the next argument in the `args` list.

The formatted string is output from left to right. When a conversion specification is encountered, the next (initially first) argument is output according to the conversion specification.

A conversion specification has the form:

```
%[flag][width][.precision][l]type
```

Each field enclosed in braces "`[ ]`" is optional and consists of a single character or number signifying a particular format option. The simplest possible conversion specification contains a percent sign (`%`) and a conversion character (ex: `%f`)

`flag`

`-` The converted argument is left-justified when printed (the default is right-justification).

`width`

digit string The numeric digit string specifies the field width for the conversion. If the converted value has fewer characters than `width`, enough blank (space) characters are output to make the total number of characters output equal the field width. The spaces are output before or after the value, depending on the presence or absence of the left-justification flag. If the field width digits have a leading zero, zeros are used as pad characters instead of spaces. If the converted string has more characters than the value of `width`, the string is truncated.

`*` The `width` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`.precision`

digit string For floating point conversions, `precision` specifies the number of digits to appear after the decimal point; for character string conversions, it specifies the maximum number of characters to be printed from a string.

`*` The `precision` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`l`

A conversion normally performed on an `int` is performed on a `long` (may be used with the `d`, `o` and `x` conversion characters).

printf( )

The type character format is as follows:

character	type of argument	output format
d	int	signed decimal
u	int	unsigned decimal
x	int	unsigned hexadecimal
o	int	unsigned octal
f	float or double	floating point
c	char	single character
s	string	character string
e	float or double	scientific nototation
g	uses d, f or e - whichever gives full precision in minimum space	

Return Values:

None

See Also:

scanf(), fprintf(), sprintf()

Example:

```
main()
{
   int i = 20;
   double x = 1.23;

   printf("The integer in decimal is %d, in octal is %o, in hex is %x.\n",i,i,i);
   printf("The double is %5.2f.\n",x);
   getch();
}
```

put_all_sks( )

Format:

```
#include <video.include>

void put_all_sks(s1, s2, s3, s4, s5, s6, s7, s8, attribute)
char *s1, *s2, *s3, *s4, *s5, *s6, *s7, *s8;   /* softkey labels (null-terminated) */
int attribute;
```

Description:

Writes the given labels onto the corresponding softkeys by calling `put_sk()` for each softkey. The softkeys are numbered from left to right. The `attribute` parameter is used to choose the video attribute for all the softkeys. The `attribute` values may be arithmetically OR'd together to create a combination of effects.

Although `INVERSE` and `INVERSE|HALF_BRIGHT` (inverse-video OR'd with low-intensity) are the attributes best suited for softkey painting, any of the attribute values listed in the `set_attribute()` description may be used. The constant values for `attribute` are defined in the `video.include` file.

To keep an already existing softkey, insert a `NULL` (`'\0'`) character in place of its label in the argument list. The function skips over the key without overwriting it. For more information on formatting individual softkeys, see the `put_sk()` description.

Return Values:

None

See Also:

`put_sk()`

put_all_sks( )

Example:

```
#include <video.include>

main()
{
   put_all_sks("s_key1",
               "s_key2",
               "s_key3",
               "s_key4",
               "s_key5",
               "s_key6",
               "s_key7",
               "s_key8",INVERSE);
   getch();
}
```

put_sk()

Format:

```
#include <video.include>      /* for attribute definitions */
#include <conio.include>      /* for SK1 - SK8 definitions */

void put_sk(label, softkey_id, attribute)
char *label;
int softkey_id, attribute;
```

Description:

Writes a label onto one of the eight softkeys using the specified display attribute. The label is centered on the softkey. This is known as "painting" a softkey.

The softkey label is a null-terminated string stored in the memory location pointed to by the `label` parameter. An opening single quote character (` -- press Shift @ on the protocol analyzer's keyboard) is used to mark the division between the upper and lower rows of the label. If the opening single quote is omitted, the label appears on the upper row of the softkey. Each row may contain up to nine (9) characters; any additional characters are truncated.

The integer constants for the `attribute` and `softkey_id` parameters are found in `video.include` and `conio.include`, respectively. The constant values for `softkey_id` and for the `attribute` options best suited for softkey painting are shown below:

attribute
NORMAL
INVERSE
BLINK
HALF_BRIGHT

softkey_id
SK1
SK2
SK3
SK4
SK5
SK6
SK7
SK8

put_sk( )

Other attribute values are shown in the set_attribute() description. The attribute options may be arithmetically "OR'd" together to create combinations of the video attributes. For example, entering INVERSE|HALF_BRIGHT as the attribute parameter creates a softkey that appears darker than a softkey created with INVERSE alone.

put_sk() returns without modifying a softkey if the label string is NULL, or if the softkey_id is not one of the SK1 through SK8 constants.

Return Values:

None

See Also:

put_all_sks()

Example:

```
#include <video.include>
#include <conio.include>

main()
{
   put_sk("This is 'S_KEY5",SK5,INVERSE|HALF_BRIGHT);
   getch();
}
```

putc( )

Format:

```
#include <stdio.include>

int putc(ch, file)
int ch;
FILE *file;
```

Description:

Writes the character `ch` to a file or output stream. The `putc` function uses the output stream specified by the `file` parameter.

Return Values:

Returns the character written (`ch`).

See Also:

`getc()`, `fputc()`

putc( )

Example:

```
#include <stdio.include>

main()
{
   char *str = "hello world";

   while(*str)
      putc(*str++,stdout);

   getch();
}
```

putch( )

Format:

```
void putch(ch)
char ch;
```

Description:

Writes the character ch to the display at the current cursor position with the current display attributes before advancing the horizontal cursor position.

Return Values:

None

See Also:

```
getch()
```

Example:

```
main()
{
   char *str = "hello world";

   while(*str)
      putch(*str++);
   getch();
}
```

puts( )

Format:

```
#include <stdio.include>

int puts(string)
char *string;
```

Description:

Writes the string pointed to by the `string` parameter to the display. `puts()` uses the current display attributes to write the string, then advances the cursor to the first column of the next line if the display is not in transparent text mode (see `set_screen_mode()`).

Return Values:

Returns the last character written, or `EOF` to indicate an error.

See Also:

```
gets()
```

Example:

```
main()
{
   puts("hello world!!!");
   getch();
}
```

rand()

Format:

```
int rand()
```

Description:

Generates a positive pseudo-random number between 0 and 32767. The `srand` function may be used to initialize a random starting point before `rand()` is called.

Return Values:

Returns a pseudo-random number as described above.

See Also:

```
srand()
```

Example:

```
main()
{
    int rand_result, i;
    double frand;

    rand_result = rand();
    printf("The random number generated is %d.\n",rand_result);
    printf("The value calculated from %d for a random number\n",rand_result);
    printf("between zero and one is %f.\n",(float)((float)rand_result/(float)32767));
    getch();
}
```

This example takes any pseudo-random numbers generated by `rand()` and converts them to floating point numbers between 0 and 1.

read( )

Format:

```
#include <stdio.include>

int read(fdesc, buff, cnt)
int fdesc;
char *buff;
unsigned int cnt;
```

Description:

Reads cnt bytes from the file whose descriptor is fdesc and stores the information in the location indicated by the buff parameter. The fdesc parameter contains an integer "handle" that was generated by the open function when the file was first encountered in the program.

Return Values:

Returns the number of bytes actually read, which may be less than cnt if there were less than cnt bytes left in the file, zero if the read function tried to begin access at the end-of-file marker, or -1 if an error occurred during function operation.

See Also:

close(), open(), write()

Example:

```
#include <stdio.include>
#include <fcntl.include>

main()
{
   int fdesc;
   long seek_result, write_result;
   char instr[20];

   if((fdesc = open("file1.text",O_WRONLY|O_CREAT)) == -1)
      printf("Can't create file1.text.");
   else
   {
      if(write_result = write(fdesc,"hello world\n",12) != 12)
         printf("Write error detected.");
      else
      {
         if((fdesc = open("file1.text",O_RDONLY)) == -1)
            printf("Can't open file1.text.");
         else
         {
            read(fdesc,instr,5);
            instr[3] = 0;
            printf("the string read after fseek is %s.",instr);
         }
      }
      close(fdesc);
   }
   getch();
}
```

read_message( )

Format:

```
#include <message.include>
#include <retval.include>

int read_message(message, wait, timeout);
MESSAGE *message;
int wait;
long timeout;
```

Description:

Reads a message from the message queue. Information from the queue is placed in the data structure pointed to by the `message` parameter.

Constant values for the `wait` parameter are defined in the `message.include` file. The choices are shown below:

`wait`	Description
`WAIT`	The function waits for a message for a period of `timeout` ticks.
`NO_WAIT`	The function doesn't wait for messages, and `timeout` is ignored.

If the `wait` parameter equals `WAIT`, `read_message()` blocks the calling process for up to `timeout` ticks (1 tick = 1/10 second) or until a message is read. If `timeout` equals `WAIT_FOREVER` (0L), the calling process is blocked until a message is read.

Return Values:

`SUCCESSFUL (0)`	A message was read.
`NO_MESSAGE (-1)`	There weren't any messages available, or `read_message()` timed out.

INVALID_QUEUE_NUMBER (-2)	The external variable _process_QID has been corrupted (see "Messaging and Event Buffer Services" in the *DataCommC Programming Language User's Guide* for more information).

See Also:

send_message(), is_msg_avail()

Example:

```
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE message;
   int message_counter = 0;

   start_data();
   while(1)
   {
      if(read_message(&message,WAIT,WAIT_FOREVER))
         printf("Error reading queue.\n");
      else
      {
         message_counter += 1;
         printf("message #%d read.\n",message_counter);
         if(message.type == DATACOMM)
            release_event(message.body.event.event_ptr);
      }
   }
}
```

read_pod_id( )

Format:

```
int read_pod_id(pod_id)
char *pod_id;
```

Description:

Reads the pod identification from the pod connected to the data link control hardware. When the function is successful, the pod id value is placed in the memory location pointed to by the `pod_id` parameter. The following `pod_id` values are located in the `dlib.include` file.

pod_id	value
RS232	0x1
V_35	0x2
RS449	0x3
X_21	0x4
MIL188C	0x5
NO_POD	0x8

Return Values:

`SUCCESSFUL (0)`

`ERROR_10 (-10)` A fatal error occurred during an attempt to communicate with the data link control hardware

Example:

```
#include <retval.include>
#include <dlib.include>

main()
{
    int result;
    unsigned pod_id;

    if((result = read_pod_id(&pod_id)) != SUCCESSFUL)
       exit();
    switch(pod_id)
    {
       case RS232:
          printf("The 4954 is connected to an RS-232 pod.\n");
          break;
       case V_35:
          printf("The 4954 is connected to a V.35 pod.\n");
          break;
       case RS449:
          printf("The 4954 is connected to an RS-449 pod.\n");
          break;
       case X_21:
          printf("The 4954 is connected to an X.21 pod.\n");
          break;
       case MIL188C:
          printf("The 4954 is connected to a MIL-188C pod.\n");
          break;
       case NO_POD:
          printf("The 4954 is not connected to a pod.\n");
          break;
       default:
          printf("The 4954 is connected to pod type %d.\n",pod_id);
    }
    getch();
}
```

read_rs232( )

Format:

```
void read_rs232(ch)
char *ch;
```

Description:

Reads a character from the RS232 port and puts it into the memory location pointed to by the ch parameter.

Return Values:

None

See Also:

```
init_rs232(), rs232_read_ready(), rs232_write_ready(), print_char(), write_rs232()
```

Example:

```
main()
{
   char ch = 0;

   init_rs232();
   printf("Reading from the RS232 port - press RESET to exit\n\n");
   while( ch != '*' )
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         printf("The character read is '%c'.\n",ch);
      }
}
```

read_vidram()

Format:

```
#include <video.include>

void read_vidram(row, column, character, attribute)
int row, column;
int *attribute;
char *character;
```

Description:

Reads the video RAM character and attribute value at the location defined by the `row` and `column` parameters. The maximum row and column values are 25 and 80, respectively. The selected character and attribute are placed in the memory locations pointed to by the `character` and `attribute` parameters, respectively.

Return Values:

None

See Also:

`write_vidram()`

read_vidram( )

Example:

```
#include <video.include>

main()
{
   char ch;
   int row = 1, col = 1;
   int attribute;

   printf("This is a test string.\n");
   read_vidram(row, col, &ch, &attribute);
   printf("The character at rol,col 1,1 is '%c'.\n",ch);
   printf("The character is being displayed with the following attributes:\n\n");

   if((attribute & NORMAL) == NORMAL)
      printf("       NORMAL\n");
   if((attribute & SPECIAL) == SPECIAL)
      printf("       SPECIAL\n");
   if((attribute & INVERSE) == INVERSE)
      printf("       INVERSE\n");
   if((attribute & HALF_BRIGHT) == HALF_BRIGHT)
      printf("       HALF_BRIGHT\n");
   if((attribute & BLINK) == BLINK)
      printf("       BLINK\n");

   if((attribute & NORMAL) == 0)
   {
      if(attribute & HEX_NORMAL)
         printf("       HEX_NORMAL\n");
      if(attribute &  HEX_INVERSE )
         printf("       HEX_INVERSE\n");
      if(attribute & HEX_BLINK )
         printf("       HEX_BLINK\n");
   }
   getch();
}
```

release_event( )

Format:

```
int release_event(event)
void *event;
```

Note The void data type is used because release_event() accepts pointers to structures of type FRAME_EVENT, LEAD_EVENT, CH_EVENT or CHE_EVENT.

Description:

Releases access to a specified event in the event buffer. The event's held_count parameter is decremented. If held_count is zero, the memory is said to be free, but it may not be resuable until all the events that are chronologically in front of the specified event have also been released by the processes that aquired access to them.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The event was not found

ERROR_2 (-2) A NULL event pointer was passed in.

ERROR_3 (-3) The specified event pointer was not in the EBS buffer range.

See Also:

get_event_bounds(), hold_event()

release_event( )

Example:

```
#include <dlib.include>
#include <retval.include>
#include <message.include>

main()
{
   MESSAGE message;
   int message_counter = 0;

   start_data();
   while(1)
   {
      if(read_message(&message, WAIT, WAIT_FOREVER))
         printf("Error reading queue.\n");
      else
      {
         message_counter += 1;
         printf("message #%d read.\n", message_counter);
         release_event(message.body.event.event_ptr);
      }
   }
}
```

resend_bops()

Format:

```
#include <retval.include>

int resend_bops(ptr, n, length)
char *ptr;
unsigned int n;
unsigned int length;
```

Description:

Retransmits the last bit-oriented protocol (HDLC, SDLC, X25, or X75) string sent using a sendf() call. If sendf() has not been called, the data held in the data link control hardware (possibly garbage) are sent out on the line. The resend_bops function allows modification of the sendf string. The sendf string may be sent many times (using resend_bops()) without redefining the string.

The ptr parameter points to a string of characters intended to replace the first n bytes of the sendf string. If ptr is NULL, or if n is zero, the sendf string is sent unchanged.

The length parameter defines the length of the new frame. If length is 0, the length remains the same as the original sendf string's length. If length is greater than the maximum allowable length (4106), length is set to 4106. If length is greater than the length of the original sendf string, the string is sent, with whatever data exist past the end of the buffer holding that string, concatenated on the end. The concatenation stops when the new string's length equals the value defined by length.

The resend_bops function overwrites the first n bytes of text in the original sendf string; it is not intended to insert text into that string.

resend_bops( )

Note Since this function is intended to provide faster traffic generation, no check is made to be sure that a bit-oriented protocol (HDLC, SDLC, X25, or X75) has been selected. Selecting a character-oriented protocol (Char Asyn/Syn or BSC in the Setup Menu, COPS or BSC with set_protocol()) with this function is not supported.

Return Values:

SUCCESSFUL (0)

ERROR_5 (-5) There were problems communicating with the data link control hardware.

ERROR_10 (-10) The data link control hardware is not running.

See Also:

sendf()

Example:

```
main()
{
   int i, result;
   char *ptr;

   ptr = "New";
   result = start_data();
   printf("result of start data is %d\n",result);
   result = sendf("Hi  there1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ");
   printf("result of sendf is %d\n",result);
   for(i=0; i<30; i++)
      result = resend_bops(ptr,3,0);  /* replace first 3 chars */

   getch();
}
```

reset_pod( )

Format:

```
int reset_pod()
```

Description:

Resets an attached pod by placing all leads in tri-state or not-driven states. Resets all pod hardware.

Return Values:

SUCCESSFUL (0)

ERROR_5 (-5) A fatal error occurred during an attempt to communicate with the protocol analyzer's data link control hardware.

See Also:

```
read_pod_id()
```

Example:

```
main()
{
   if (reset_pod())
      printf("Error resetting the 4954 pod.\n");
   else
      printf("The pod is reset.\n");

getch();
}
```

restore_cursor( )

Format:

```
void restore_cursor()
```

Description:

Restores the cursor to the screen position saved during the last call to `save_cursor()`.

Return Values:

None

See Also:

```
save_cursor(), set_cursor(), get_cursor()
```

Example:

```
main()
{
   int row = 2, col = 3;

   set_cursor(row, col);
   save_cursor();
   printf("Press a key: ");
   getch();
   restore_cursor();
   printf("Now press a key to exit: ");
   getch();
}
```

resync( )

Format:

```
#include <retval.include>

int resync()
```

Description:

Causes the protocol analyzer, when a character-oriented protocol is selected, to drop sync and begin looking for sync characters to re-establish sync. The data link control hardware must be running from a `start_data()` call before this function is called. The resync mode should be set to `MANUAL` with `set_resync()` before using this function

---

Note Due to speed considerations, no check is made for a valid character-oriented protocol (`Char Asyn/Syn` or `BSC` when set in the Setup Menu, `COPS` or `BSC` when `set_protocol()` is used) during this call, nor is a check made for resync mode equal to `MANUAL`.

Return Values:

`SUCCESSFUL` (0)

`ERROR_5` (-5) There were problems communicating with the data link control hardware.

`ERROR_10` (-10) The data link control hardware is not running.

See Also:

`set_resync()`

Example:

```
#include <retval.include>

main()
{
    int result;

    start_data();
    result = resync();
    if(result != SUCCESSFUL)
       printf("The resync failed with a value of %d\n", result);
    else
       printf("The resync was successful\n");

    getch();
}
```

roll_down( )

Format:

```
void roll_down(start, end, num_lines)
int start, end, num_lines;
```

Description:

Rolls lines of text toward the bottom of the screen. The area affected by the scroll is bounded by the rows indicated by the `start` and `end` parameters. This text area is rolled down by the number of rows specified in `num_lines`.

Return Values:

None

See Also:

```
roll_up()
```

Example:

```
main()
{
   set_cursor(10,1);
   printf("hello world");
   while(1)
   {
      if((tolower(getch())) == 'd')
         roll_down(1,25,1);
      else
         roll_up(1,25,1);
   }
}
```

roll_up()

Format:

```
void roll_up(start, end, num_lines)
int start, end, num_lines;
```

Description:

Rolls lines of text toward the top of the screen. The area affected by the scroll is bounded by the rows indicated by the `start` and `end` parameters. This text area is rolled up by the number of rows specified in `num_lines`.

Return Values:

None

See Also:

`roll_down()`

Example:

```
main()
{
   set_cursor(10,1);
   printf("hello world");
   while(1)
   {
      if((tolower(getch())) == 'd')
         roll_down(1,25,1);
      else
         roll_up(1,25,1);
   }
}
```

rs232_lock()

Format:

```
void rs232_lock()
```

Description:

Returns when the RS232 port has been locked by the calling process. If the port had been previously locked by another process, the calling process is blocked until it can lock the port.

Return Values:

None

See Also:

```
rs232_unlock()
```

Example:

```
main()
{
   char ch = 0;

   init_rs232();
   rs232_lock();
   while( ch != '*' )
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         printf("The character read is '%c'.\n",ch);
      }
   rs232_unlock();
}
```

rs232_read_ready( )

Format:

```
int rs232_read_ready()
```

Description:

Checks the status of the RS232 receive register. When the receive register is ready, the `read_rs232` function can be used to receive a character from the device connected to the port. If more than one process is using the port, the `rs232_lock` function should be called before any RS232 functions are called, and `rs232_unlock()` should be called after the process is finished with the port.

As with all the RS232 functions, this function must not be used until `init_rs232()` has been called.

Return Values:

Returns a 1 if a character is waiting in the receive register, or zero if there aren't any characters waiting in the register.

See Also:

`rs232_write_ready()`, `rs232_lock()`, `rs232_unlock()`, `read_rs232()`

rs232_read_ready()

Example:

```
main()
{
   char ch = 0;

   init_rs232();
   while( ch != '*' )
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         printf("The character read is '%c'.\n",ch);
      }
}
```

rs232_unlock( )

Format:

```
void rs232_unlock()
```

Description:

Releases control of the RS232 port so that another process may lock and use the port. `rs232_unlock()` only works when called by the process that locked the port with `rs232_lock()`.

Return Values:

None

See Also:

```
rs232_lock()
```

Example:

```
main()
{
   char ch = 0;

   init_rs232();
   rs232_lock();
   while( ch != '*' )
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         printf("The character read is '%c'.\n",ch);
      }
   rs232_unlock();
}
```

rs232_write_ready()

Format:

```
int rs232_write_ready()
```

Description:

Checks the status of the RS232 transmit register. When the transmit register is ready, the `write_rs232` function can be used to send a character to the device connected to the port. If more than one process is using the port, the `rs232_lock` function should be called before any RS232 functions are called, and `rs232_unlock()` should be called after the process is finished with the port.

As with all the RS232 functions, this function must not be used until `init_rs232()` has been called.

Return Values:

Returns a 1 if the transmit register is empty, and returns zero otherwise.

See Also:

`rs232_read_ready()`, `write_rs232()`, `rs232_lock()`, `rs232_unlock()`

Example:

```
#include <conio.include>

/* A dumb terminal emulator, terminates after the '~' character is typed */
/* on the keyboard....                                                   */

main()
{
   char ch = 0;

   init_rs232();
   printf("Writing to the RS232 port - press RESET to exit\n\n");
   while( ch != '~' )
   {
      if(is_key_avail())
      {
         ch = getch();
         while (!rs232_write_ready()) ;
         write_rs232(ch);
      }
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         putch(ch);
         ch = 0;
      }
   }
}
```

save_cursor( )

Format:

```
void save_cursor()
```

Description:

Saves the current cursor position so that the row and column values can be restored by `restore_cursor()`.

Return Values:

None

See Also:

```
restore_cursor()
```

Example:

```
main()
{
    int row = 2, col = 3;

    set_cursor(row, col);
    save_cursor();
    printf("Press a key: ");
    getch();
    restore_cursor();
    printf("Now press a key to exit: ");
    getch();
}
```

scanf( )

Format:

```
int scanf(format, args)
char *format;
char *args;
```

Description:

Takes text characters from the standard input stream (`stdin`), checks the character types against conversion characters imbedded in a control string pointed to by the `format` parameter, and places matching text in the fields pointed to by the `args` list.

The following `scanf()` example shows the main components of the function:

```
scanf("%f%s", &fltptr, &strptr);
```

The string `"%f%s"` is a control string (`format`) with two control items (both conversion characters), one indicating a floating point value (`%f`), and the other (`%s`) indicating a string. The other arguments, `&fltptr` and `&strptr`, define the argument list (`args`). If `scanf()` finds floating point characters, it places them in the memory location pointed to by `fltptr`; if a character string is found next, it is placed in the memory location pointed to by `strptr`.

A control string contains these control items:

- Conversion specifications
- Optional white space characters (tab, space, newline)
- Optional alphanumeric characters (not white space, and not part of a conversion specification).

The `scanf` function works its way through a control string from left to right, trying to match each control item to a portion of the input stream. During the matching process, `scanf()` fetches characters one at a time from the input stream. If a character is found which doesn't match the type specifier for the corresponding conversion specificiation, `scanf()` pushes the character back onto the input stream and finishes processing the current control item. This "pushing back" frequently gives unexpected results when a stream is used later by other I/O functions, such as `getc()`, as well as by `scanf()` itself, if it is used again.

scanf( )

A conversion specification has the form:

```
%[*] [width] [l] type
```

Each field enclosed in braces "[ ]" is optional and consists of a single character or number signifying a particular format option. The simplest possible conversion specification contains a percent sign (%) and a type, or conversion character (ex: %f)

The optional fields are defined below, and conversion characters are discussed later in this segment.

*

Assignment suppression character. The current stream is scanned, but not saved. The function goes on to the next control string item.

width

This field specifies the maximum number of characters to be fetched for the conversion.

l

This field indicates that the argument is a pointer to a long data type - the exact type (for example, long decimal, long hex, long unsigned) is determined by the conversion character.

The conversion character format is as follows:

character	type of argument	expected input format
d	pointer to int	signed decimal
u	pointer to int	unsigned decimal
x	pointer to int	unsigned hexadecimal
o	pointer to int	unsigned octal
e,f	pointer to float	floating point
c	pointer to char	single character
s	pointer to char array	character string

When a conversion specification is encountered in the control string, the scanf function skips leading white space on the input stream, then collects characters from the stream until it encounters one that is not appropriate for the corresponding conversion character. That character is pushed back onto the input string.

As long as the conversion specification didn't request assignment suppression (see '*', above), the text string that was read from the keyboard is converted to the format specified by the conversion specification, the result is placed in the location pointed to by the corresponding args argument, the next argument becomes current, and the function proceeds to the next control string item.

If assignment suppression was requested, the `scanf` function ignores the input characters and goes on the the next control item.

If an ordinary character is found in the control string, outside any conversion specfication, `scanf()` fetches the next character. If that character matches the character in the control string, the function goes on to the next control string item, ignoring the input character. If there is no match, `scanf()` terminates.

If a white space character is found in the control string, the `scanf` function fetches input characters until the first non-white space character is read. The non-white space character is pushed back onto the input stream and `scanf()` proceeds to the next item in the control string.

Return Values:

Returns the number of items converted and assigned to memory locations in `args`. Unmatched items, since they are not assigned to `args`, are not included in the count.

See Also:

```
fscanf(), sscanf()
```

scanf( )

Example:

```
#include <stdio.include>

main()
{
   char instr[70];

   scanf("%s",instr);
   printf("The string read from the input is %s.", instr);
   getch();
}
```

send_message()

Format:

```
#include <message.include>
#include <retval.include>

int send_message(QID, message, options);
long QID;
MESSAGE *message;
int options;
```

Description:

Sends a message to the queue identified by the QID (queue identifier) parameter. The options parameter indicates whether or not additional action is necessary when the message is sent. Constant values for the options parameter are defined in the message.include file and are outlined below:

options	Description
ZERO	No addition action is taken. The message is processed normally.
EXPEDITE	The message is placed at the beginning of the destination process's message queue.

send_message() provides communication between processes running in the DataCommC environment. The function expects that the process associated with the specified queue is running, and that the QID for that process queue has not been changed.

Return Values:

SUCCESSFUL (0)	The message was sent.
QUEUE_DELETED (-1)	The queue has been deleted

send_message( )

`INVALID_QUEUE_NUMBER (-2)`	An invalid queue number was given.
`NO_QUEUE_BUFFERS (-5)`	There weren't any system buffers available.
`MAX_MSGS_EXCEEDED (-6)`	The queue was full.

See Also:

`read_message(), is_msg_avail()`

Example:

```
#include <system.include>
#include <message.include>
#include <retval.include>
/* Sends an exit message to the process that spawned it.  send_message() returns an */
/* error if the parent is no longer running, the _parent_QID variable was modified, */
/* or if the process calling the exit_process() routine has been started by the */
/* run-time  routine. */

main()
{
   printf("Press a key to send parent the exit message and exit this process:\n");
   getch();
   exit_process(1);
}

/* Make the message type unique so the parent process doesn't mistake it for another */
#define EXIT_USER_PROCESS  25

exit_process(exit_code)
int exit_code;
{
   MESSAGE msg;

   msg.type = EXIT_USER_PROCESS;
   msg.body.user_message.user_long  = _process_QID;
   msg.body.user_message.user_int1 = exit_code;
   if(send_message(_parent_QID,&msg,0))
      {
      printf("Error sending exit message to parent. Press a key to exit: ");
      getch();
      }
   exit();
}
```

sendf( )

Format:

```
#include <retval.include>
#include <dlib.include>

int sendf(format, args);
char *format;
char *args;
```

Description:

Sends a formatted string to the line (by way of the protocol analyzer's level 1 control hardware). The formatting is controlled by the arguments referred to above, and is similar to the formatting shown in the `printf()` and `scanf()` functions. The `format` parameter represents a control string containing format conversion characters, and `args` represents an argument list. The control string is always null-terminated. For example:

```
sendf("%f%2n%s", BAD_FCS, 0x00, 0x04, "L2 data field");
```

In this example, `"%f%2n%s"` is a control string that sends a frame check sequence (`BAD_FCS`),two 8-bit hex values (`0x00` and `0x04`), and a character string (`"L2 data field"`).

The control string is read left to right. As each control string element is read, the character specification it contains is used to modify the corresponding field(s) in the argument list. Any character read that is not part of a character specification is translated to the current datacode and parity before being sent (for example, `"hello world"` in the control string: `"%nhello world%s"`).

A character specification has the form:

```
%[dimension]conversion character
```

The dimension field is optional and may refer to either the number of argument list fields , the value of an argument, or the length of a field to be sent, depending on which conversion character it is modifying. The dimension field consists of either a constant or an asterisk (*) preceding the conversion character. If an asterisk precedes the conversion character, the next

field in the argument list is interpreted as the integer length or number of arguments to be sent. For example, the sendf() statement below is the same as the one shown on the previous page, except that the number of 8-bit values to be sent is now included in the argument list.

```
sendf("%f%*n%s", BAD_FCS, 2, 0x00, 0x04, "L2 data field");
```

The conversion character format is as follows:

character	type of argument	output format
n	integer	8-bit value
b	pointer to buffer	hex data dump
c	frame_type	variable (?)
s	string	character string
f	FCS type	hex value

n This conversion character allows 8-bit values to be sent to the protocol analyzer's data link control hardware. The dimension field determines how many 8-bit values appear in the final send sequence. The ascii.include and ebcdic.include files contain constant values and definitions for characters in the standard ASCII and EBCDIC character sets.

b This conversion character provides a pointer to a buffer of hex data. The dimension field specifies the length of the buffer. The buffer pointer must be included in the appropriate place in the argument list. For example:

```
char *buff_ptr;          /* set up buffer pointer */
.
.
sendf("%32b", buff_ptr);
```

c The HP 18356A ISDN Solution uses this conversion character to select either the primary or secondary channel for data transmission through sendf(). If the dimension field is 1, the primary channel is selected, if it is 2, the secondary channel is selected. This conversion character must be used in combination with another conversion character that formats or specifies the data to be sent.

s This conversion character creates an ASCII (null-terminated) string. The data will be translated to the current datacode and parity (see set_protocol() function). The dimension field can be used to set an absolute length for the string, so that null characters can be included in the string. If the dimension field is not used, the string length is calculated to be the number of characters found before a null character is detected.

f This conversion character adds the specified FCS (frame check sequence) to the send string whenever "%f" is found in the control string, and a corresponding FCS parameter in an argument list field. If there is no "%f" element in the control string, GOOD_FCS is sent. The constant values are:

GOOD_FCS	Good FCS
BAD_FCS	Bad FCS
ABORT_FCS	ABORT/BREAK sequence
NO_FCS	No FCS sent (only for BSC protocol)

These constant values are defined in the dlib.include file.

Note The send string must always contain at least 2 bytes of data.

Return Values:

SUCCESSFUL (0)

ERROR_1	(-1)	An error was found in the formatting.
ERROR_2	(-2)	The data link control hardware could not send this string.
ERROR_3	(-3)	Invalid FCS type specified.
ERROR_4	(-4)	Illegal frame type encountered.
ERROR_5	(-5)	An error occurred during an attempt to communicate with the data link control hardware.
ERROR_6	(-6)	The dlc hardware is not configured to transmit.
ERROR_7	(-7)	A BOPS send string was greater than 4106 bytes or less than 2 bytes, or a COPS send string was greater than 4106 bytes or less than 1 byte.
ERROR_10	(-10)	The dlc hardware is not running.

See Also:

```
start_data()
```

Example:

```
#include <dlib.include>
#include <message.include>
#include <retval.include>

main()
{
    char buffer[10];
    int i;
    MESSAGE msg;

    init_datacomm();
```

```
    for (i = 0; i < 500; i++)
    {
        sprintf(buffer,"%d", i);
        set_cursor(10, 10);
        printf("buffer = %s, length = %d", buffer, strlen(buffer));
        sendf("%s(%n)",buffer,strlen(buffer));
        while(is_msg_avail() )
            read_message(&msg, WAIT, WAIT_FOREVER);
        release_event(msg.body.event.event_ptr);
    }
    stop_data();
}

/* Initalize the 4954 as a DTE */

init_datacomm()
{
    int result, ch;

    result = set_channelconfig(DTE, TX);
    if(result != 0)
        printf("config(DTE, TX) error - result = %d\n", result);
    result = set_channelconfig(DCE, RX);
    if(result != 0)
        printf("config(DCE, RX) error - result = %d\n", result);
    set_buffer_sizes(50, 50);
    result = start_data();
    if(result != 0)
        printf("start_data() error - result = %d\n", result);
}
```

set_attribute( )

Format:

```
#include <video.include>

void set_attribute(attribute)
int attribute;
```

Description:

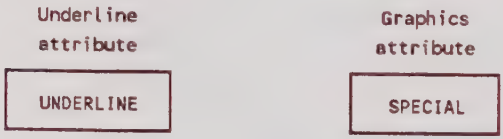
Sets the value of the current display attribute. The constant values defined in the `video.include` file generate normal, inverse-video or low-intensity text, as well as text that blinks for either the ASCII, Hex or EBCDIC character set.

ASCII attribute	Hex attribute	EBCDIC attribute
NORMAL	HEX_NORMAL	EBCDIC_NORMAL
INVERSE	HEX_INVERSE	EBCDIC_INVERSE
BLINK	HEX_BLINK	EBCDIC_BLINK
HALF_BRIGHT	HEX_HALF_BRIGHT	EBCDIC_HALF_BRIGHT

These attribute values may be set individually or combined using an arithmetic OR statement. The example program selects low-intensity, inverse-video characters.

set_attribute( )

Setting the attribute to SPECIAL enables the display of graphics characters from the currently selected display bank. For complete listings of the graphics character sets in the protocol analyzer's four display banks, refer to appendix A of this manual, "Video Character Sets."



Once the display attribute is defined, it retains its new value until set_attribute() is called again. Any text entered with putc(), putch(), puts(), center() or printf() is generated using the current attribute.

Return Values:

None

See Also:

get_attribute(), set_disp_bank()

Example:

```
#include <video.include>

main()
{
   int attribute;

   set_attribute(INVERSE | HALF_BRIGHT);
   attribute = get_attribute();
   printf("The following screen attributes are set:\n\n");
   if((attribute & NORMAL) == NORMAL)
      printf("       NORMAL\n");
   if((attribute & SPECIAL) == SPECIAL)
      printf("       SPECIAL\n");
   if((attribute & INVERSE) == INVERSE)
      printf("       INVERSE\n");
   if((attribute & HALF_BRIGHT) == HALF_BRIGHT)
      printf("       HALF_BRIGHT\n");
   if((attribute & BLINK) == BLINK)
      printf("       BLINK\n");
   if((attribute  &  NORMAL) == 0)
   {
      if(attribute & HEX_NORMAL)
         printf("       HEX_NORMAL\n");
      if(attribute & HEX_INVERSE )
         printf("       HEX_INVERSE\n");
      if(attribute & HEX_BLINK )
         printf("       HEX_BLINK\n");
   }
   getch();
}
```

set_bcc( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_bcc(startbcc1, startbcc2, stopbcc1, stopbcc2)
char startbcc1, startbcc2, stopbcc1, stopbcc2;
```

Description:

Sets the protocol rules for calculating the block check character (bcc) as long as the protocol has been set to COPS and the errorcheck is any available value except NO_ERROR_CHECK (see set_protocol()).

The startbcc1 and startbcc2 parameters are used to start bcc accumulation. Accumulation begins when either of these two values are received. Similarly, when either the stopbcc1 or stopbcc2 parameter values are received, bcc accumulation ends.

Since set_bcc() affects the protocol analyzer's data link control setup, it must be called only when the dlc hardware is not active, that is, either before start_data() is called, or after stop_data() is called.

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The protocol was not COPS.

ERROR_2 (-2) The errorcheck was NO_ERROR_CHECK.

ERROR_10 (-10) The data link control hardware has already been started. The dlc hardware must be stopped with stop_data() before any changes to the dlc configuration can be made.

set_bcc( )

See Also:

get_startbcc(), get_stopbcc(), set_protocol()

Example:

```
#include <ascii.include>      /* for as_STX, as_SOH, as_ETX and as_EOT */
#include <retval.include>     /* for SUCCESSFUL */

main()
{
   int startbcc1 = as_STX,
       startbcc2 = as_SOH,
       stopbcc1  = as_ETX,
       stopbcc2  = as_EOT;
   int result;

   if((result = set_bcc(startbcc1, startbcc2, stopbcc1, stopbcc2)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_bcc().\n",result);
      getch();
      exit();
   }

   printf("Start and stop characters for the BCC are set.");
   getch();
}
```

set_bitrate( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_bitrate(bitrate)
long bitrate;
```

Description:

Sets the bitrate for the clock provided on the HP 4954 Protocol Analyzer. This function also affects the resolution of the timestamp on the data. If the value selected is less than the minimum value (10 bps), the bitrate defaults to the minimum value. Similarly, if the bitrate valued selected is larger than the maximum value (256k bps), the bitrate defaults to the maximum value. Nominal performance is not guaranteed when the bitrate is greater than or equal to 100k bps. `set_bitrate()` affects the protocol analyzer's data link control setup; any changes must be made either before the dlc hardware is started with `start_data()`, or after `stop_data()` has halted the dlc.

Return Values:

`SUCCESSFUL (0)`

`WARNING_1 (1)` The value of the `bitrate` parameter exceeded the maximum allowable value (256k bps); the maximum value is substituted.

`WARNING_2 (2)` The value of the bitrate parameter is less than the minimum allowable value (10 bps); the minimum value is substituted.

`ERROR_10 (-10)` The data link control hardware has already been started. The dlc hardware must be stopped with `stop_data()` before any changes to the dlc configuration can be made.

set_bitrate( )

See Also:

```
get_bitrate()
```

Example:

```
main()
{
   long bitrate;

   set_bitrate(1200L);
   bitrate = 0;
   get_bitrate(&bitrate);
   printf("The bitrate is %ld.",bitrate);
   getch();
}
```

set_buffer_sizes( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_buffer_sizes(DTE_size, DCE_size)
int DTE_size, DCE_size;
```

Description:

Sets the sizes for the primary DTE and DCE data buffers and sets the secondary buffers (SDTE and SDCE) sizes to zero. The primary buffer sizes (`DTE_size` and `DCE_size`) must be defined as percentages of the total buffer space. The total buffer space is 205676 bytes.

If `start_data()` has already been called, and the data link control hardware is running when `set_buffer_sizes()` is called, both the DTE and DCE buffer sizes are set to 50 percent of the total buffer (102838 bytes each).

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` One or both of the percentages were less than zero.

`ERROR_2 (-2)` One or both of the percentages were greater than one hundred.

`ERROR_3 (-3)` The combined percentages were greater than one hundred.

See Also:

`start_data()`

set_buffer_sizes( )

Example:

```
#include <retval.include>

main()
{
    int result;

    if((result = set_buffer_sizes(75, 25)) != SUCCESSFUL)
    {
        printf("Error #%d returned from set_buffer_sizes().\n", result);
        getch();
        exit();
    }
    printf("Buffer sizes set.");
    getch();
}
```

set_channelconfig( )

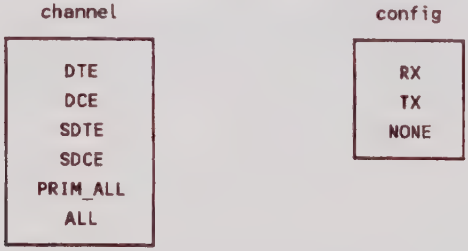
Format:

```
#include <dlib.include>
#include <retval.include>

int set_channelconfig(channel, config)
char channel, config;
```

Description:

Sets the configuration of the specified channel. Any of the protocol analyzers' channels may be set to receive data or to idle, but only one may be selected to transmit. Constant values for the channel and config parameters are defined in the dlib.include file and are shown below:



The channels include the primary DTE and DCE (DTE and DCE) and the secondary DTE and DCE (SDTE and SDCE). The secondary channels are intended for use with the HP 18356A ISDN Solution. All four channels may be set to receive data if channel=ALL and config=RX; both primary channels may be selected to receive if channel=PRIM_ALL and config=RX.

If channel is set to PRIM_ALL or ALL, the config parameter must be RX.

Note Only one channel can be configured as a transmitter at a time. If you call `set_channelconfig()` requesting transmit capability when a different channel has already been set up to transmit, the function terminates in error.

The `channel/config` setup determines whether the protocol analyzer acts in monitor or simulate mode, and whether the Event Buffer Service (EBS) is active. Whenever any channel is selected to receive data, the Event Buffer begins collecting data (after `start_data()` is called). If a channel is selected to transmit, simulate mode is enabled. If the transmitter is not enabled, the protocol analyzer is configured for monitor mode.

The table below shows the possible combinations for the primary channels (combinations for the secondary channels are similar). In the table, the state of the protocol analyzer is represented by the boxed text at the intersection of the DTE and DCE channel configurations. For example, when `DTE==RX` and `DCE==TX`, the protocol analyzer is in simulate mode because a transmitter is enabled, and the Event Buffer is ready to collect data because a receiver is enabled.

		DTE		
		RX	TX	NONE
DCE	RX	monitor EBS	simulate EBS	monitor EBS
	TX	simulate EBS	ERROR_1	simulate
	NONE	monitor EBS	simulate	monitor

Return Values:

`SUCCESSFUL (0)`

set_channelconfig( )

WARNING_1 (1)	The value of one or more of the parameter values is unknown. All four channels are set to RX.
ERROR_1 (-1)	Transmit (TX) was selected while transmit capability had already been selected for another channel.
ERROR_2 (-2)	The channel parameter was set to ALL or PRIM_ALL but config was not set to RX.
ERROR_10 (-10)	The data link control hardware has already been started. The dlc hardware must be stopped with stop_data() before any changes to the dlc configuration can be made.

See Also:

get_channelconfig()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int result;

   if((result = set_channelconfig(DTE, RX)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_channelconfig().\n", result);
      getch();
      exit();
   }
   printf("DTE channel configured to receive.");
   getch();
}
```

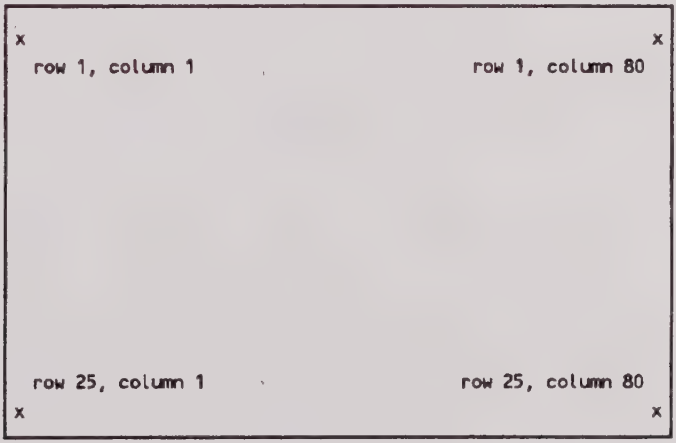
set_cursor( )

Format:

```
void set_cursor(row, column)
int row, column;
```

Description:

Places the cursor on the row and column indicated. The screen boundaries are shown below:



Return Values:

None

See Also:

```
get_cursor()
```

set_cursor( )

Example:

```
main()
{
    set_cursor(15, 35);
    printf("This string starts at screen coordinates (15, 35).");
    getch();
}
```

set_data_source( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_data_source(data_source, data_file)
char data_source;
char *data_file;
```

Description:

Allows a program to set either the line or a disc file as the source for the data buffer input. The `set_data_source` function must be called before the data link control hardware is started when the program is going to run from a disc file.

`DISC_RUN` allows data to be brought in from the file indicated by the `data_file` parameter. Files resident on any of the protocol analyzers' internal or attached disc drives may be brought in by specifying the device name in the `data_file` parameter. See the *DataCommC Programming Language User's Guide* for a list of possible disc drive device names. Hard discs may also include subdirectory names in the path. For example, "`c:C/User/Mydata.bufferdata`" could be a valid path/file name for the protocol analyzer's internal hard disc if the file "`Mydata.bufferdata`" exists under the "`C/User`" subdirectory. The file indicated by `data_file` must exist and have a `bufferdata`, `menus&data` or `ext.rundata` extension.

The `data_file` parameter is ignored if `data_source==LINE_RUN`. When `data_source==DISC_RUN`, the file is opened automatically, and is closed when `stop_data()` is called. Also when `data_source==DISC_RUN`, the protocol analyzer is automatically configured for monitoring; it is not possible to send data or set leads.

If the source of data has been set to `DISC_RUN`, `set_data_source()` must be called before each `start_data()` call.

Return Values:

`SUCCESSFUL (0)` The operation was successful.

set_data_source( )

WARNING_1 (1) The value of one or more of the parameter values is unknown. It is assumed to be the default. The default for the data_file parameter is NULL.

ERROR_1 (-1) The file specified by data_file does not exist, or the file extension is something other than bufferdata, menus&data or ext.rundata.

See Also:

get_data_source(), start_data(), stop_data()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
    int result;
    char *runfile;

    runfile = "c:run010189.menus&data";
    if((result = set_data_source(DISC_RUN, runfile)) != SUCCESSFUL)
    {
        printf("Error #%d returned from set_data_source().\n", result);
        getch();
        exit();
    }
    printf("The 4954 is configured to run from disk file '%s'.", runfile);
    getch();
}
```

set_disp_bank( )

Format:

```
#include <video.include>

void set_disp_bank(disp_bank)
int disp_bank;
```

Description:

Selects one of four special display banks resident in the protocol analyzer. The display banks contain graphics character sets that may be used to create bar charts for data, boxes to offset windowed text, or japanese characters. Tables containing the display bank characters are included in appendix A, "Video Character Sets," in this manual. Constant values for the `disp_bank` parameter are defined in the `video.include` file, and are shown below:

disp_bank	value
DISP_BANK0	0x000
DISP_BANK1	0x100
DISP_BANK2	0x200
DISP_BANK3	0x300

The `set_display_bank` function has no effect unless the display attribute has been set to `SPECIAL`. This can be accomplished by calling:

```
set_attribute(SPECIAL);
```

before calling `set_disp_bank()`. See `set_attribute()` for more information.

The display banks can also been seen by pressing **Ext Test/ Set Clock** from the protocol analyzer's Top Level Menu, then selecting **Display Patterns**. Any of the four banks may be viewed by pressing the appropriate softkey.

set_disp_bank( )

Return Values:

None

See Also:

get_attribute(), set_attribute()

Example:

```
#include <video.include>

main()
{
    int x, z;

    set_attribute(SPECIAL);
    set_disp_bank(DISP_BANK0);
    for(x = 128; x < 256; x++)
    {
        putch(x);
        printf("  ");
        z = (x - 7) % 24;
        if(!z)
            printf("\n\n");
    }
    getch();
}
```

set_duplex( )

Format:

```
#include <dlib.include>

int set_duplex(duplex)
int duplex;
```

Description:

Modifies the duplex value for the conversation between DTE and DCE devices by passing a constant value through `duplex` parameter. The constant value may be either:

duplex	value
HDX	1
FDX	2

These values are located in the `dlib.include` file.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` An invalid parameter was passed to the function.

`ERROR_10 (-10)` The data link control hardware has already been started. The dlc hardware must be stopped with `stop_data()` before any changes to the dlc configuration can be made.

See Also:

`get_duplex()`

set_duplex( )

Example:

```
#include <dlib.include>
#include <retval.include>
main()
{
    int result;

    result = set_duplex(FDX);
    if (result != SUCCESSFUL)
    {
        printf("set_duplex() failed with a value of %d\n",result);
    }
    else
    {
        printf("There was a problem setting the duplex\n");
    }
    getch();
}
```

set_error_handler( )

Format:

```
void set_error_handler(error_QID)
long error_QID;
```

Description:

Designates `error_QID` as the queue to handle non-fatal errors. If `error_QID` equals zero, the default system error handler is used.

Return Values:

None

See Also:

`error_notification()`

set_error_handler( )

Example:

```
#include <message.include>
#include <system.include>

#define The_sun_sets_in_the_west 1
main()
{
   MESSAGE message;
   int error_count = 0;

   set_error_handler(_process_QID); /* tell the run-time system that this */
                                    /* process will handle all non-fatal  */
                                    /* errors                             */
   while( The_sun_sets_in_the_west )
   {
      read_message(&message,WAIT,WAIT_FOREVER);
      if(message.type == ERROR)
         if(++error_count == 10) /* exit run-time environment if more than */
                                 /* 10 non-fatal errors are detected       */
            error_notification("Error limit exceeded, exiting to shell..",FATAL_ERROR);
   }
}
```

set_ignored_leads( )

Format:

```
int set_ignored_leads(lead_id_map)
unsigned long lead_id_map;
```

Description:

Programs the protocol analyzer's data link control hardware to ignore transitions on leads that are bit-mapped in `lead_id_map` with a 1. Lead 0 (`0x0001`) corresponds to bit 0 of `lead_id_map`, lead 1 (`0x0002`) corresponds to bit 1, and so on. Only leads 0 through 14 (`0x4000`) can be chosen to be ignored. See the `leads.include` file for definitions of the lead constants. The leads and their constant values are also shown below. If `lead_id_map` = `0x0000`, none of the leads are ignored.

RS232C/MIL188C

lead	value
RTS	0x0001l
CTS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
SQ	0x0040l
DRS	0x0080l
SRS	0x0100l
SCS	0x0200l
SCD	0x0400l

RS449

lead	value
RS	0x0001l
CS	0x0002l
DM	0x0004l
TR	0x0008l
IC	0x0010l
RR	0x0020l
SQ	0x0040l
SI	0x0080l
SRS	0x0100l
SCS	0x0200l
SRR	0x0400l
IS	0x0800l
SF	0x1000l
RL	0x2000l
SS	0x4000l

V.35

lead	value
RS	0x0001l
CS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
LT	0x0040l

set_ignored_leads( )

Return Values:

SUCCESSFUL (0)

ERROR_10 (-10) The data link control hardware was already running, and the leads could not be configured. Use `stop_data()` to halt the dlc hardware before `set_ignored_leads()` is called.

See Also:

`get_ignored_leads()`

Example:

```
#include <leads.include>
#include <retval.include>

main()
{
    long leads = DTE_LEAD | RTS;
    int result;

    if((result = set_ignored_leads(leads)) != SUCCESSFUL)
    {
        printf("Error #%d returned from set_ignored_leads().\n",result);
        getch();
        exit();
    }
    printf("The leads to ignore have been set.");
    getch();
}
```

set_lead( )

Format:

```
#include <leads.include>
#include <dlib.include>
#include <retval.include>

int set_lead(lead_id, lead_state)
unsigned long lead_id;
unsigned int lead_state;
```

Description:

Sets the lead specified by `lead_id` to the state (`ON`, `OFF` or `NOT_DRIVEN`) defined in `lead_state`. The `lead_id` parameter refers to leads defined by RS232C, MIL188C, RS449 or V.35 interface standards. Valid `lead_id` definitions are located in the `leads.include` file and are shown below:

RS232C/MIL188C

lead_id	value
RTS	0x0001l
CTS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
SQ	0x0040l
DRS	0x0080l
SRS	0x0100l
SCS	0x0200l
SCD	0x0400l

RS449

lead_id	value
RS	0x0001l
CS	0x0002l
DM	0x0004l
TR	0x0008l
IC	0x0010l
RR	0x0020l
SQ	0x0040l
SI	0x0080l
SRS	0x0100l
SCS	0x0200l
SRR	0x0400l
IS	0x0800l
SF	0x1000l
RL	0x2000l
SS	0x4000l

V.35

lead_id	value
RS	0x0001l
CS	0x0002l
DSR	0x0004l
DTR	0x0008l
RI	0x0010l
CD	0x0020l
LT	0x0040l

set_lead( )

Return Values:

SUCCESSFUL (0)

ERROR_1 (-1) The lead_id definition is invalid.

ERROR_2 (-2) The lead_state definition is invalid.

ERROR_3 (-3) The protocol analyzer is in monitor mode and leads cannot be set (see set_channelconfig()).

ERROR_5 (-5) Problems occurred while attempting to communicate with the protocol analyzer's data link control hardware.

See Also:

get_lead(), set_ignored_leads(), get_ignored_leads()

Example:

```
#include <dlib.include>
#include <leads.include>
#include <retval.include>

main()
{
   int result;

   if((result = start_data()) != SUCCESSFUL)
   {
      printf("Error #%d returned from start_data().\n",result);
      getch();
      exit();
   }
   if((result = set_lead(RTS,ON)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_lead().\n",result);
      getch();
      exit();
   }
   printf("The RTS lead is now on.");
   getch();
}
```

set_lead_control( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_lead_control(lead_control)
int lead_control;
```

Description:

Modifies the lead_control value for the conversation between the DTE and DCE devices by passing a constant value through the `lead_control` parameter. The constant value is either:

lead_control	value
LEAD_CONTROL_DEFAULT	2
LEAD_CONTROL_USER_DEF	1

These `lead_control` values are located in the `dlib.include` file.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` An invalid parameter was passed to the function.

`ERROR_10 (-10)` The data link control hardware has already been started. The dlc hardware must be stopped with `stop_data()` before any changes to the dlc configuration can be made.

See also:

`start_data(), get_lead_control()`

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
    int result;
    result = set_lead_control(LEAD_CONTROL_DEFAULT);
    if (result != SUCCESSFUL)
    {
        printf("set_lead_control() failed with a value of %d\n",result);
    }
    else
    {
        printf("There was a problem setting the lead control\n");
    }
    getch();
}
```

set_protocol( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_protocol(protocol, mode, datacode, parity, errorcheck)
int protocol, mode, datacode, parity, errorcheck;
```

Description:

Modifies certain datacomm parameters (protocol, mode, datacode, parity, and errorcheck) that are otherwise only accessible through the protocol analyzer's Setup menu. The DataCommC Programming Language supports bit-oriented and synchronous and asynchronous character-oriented protocols. Choices for the protocol parameter include any of the following:

protocol	
HDLC	High-level Data Link Control
SDLC	Synchronous Data Link Control
X25	X.25 - Level 3 Bit-oriented Protocol
X75	X.75 - Level 3 Bit-oriented Protocol
COPS	Character-Oriented Protocols
BSC	Bisync
NO_CHANGE	Not a protocol - allows old choice to remain unchanged

The choices for mode, datacode, parity, and errorcheck depend on the protocol chosen, and lists of the choices available for each protocol are shown on the following pages. If start_data() is called, the values selected for the set_protocol() parameters are downloaded to the protocol

analyzer's data link control hardware, which uses them to interpret received data and to translate outgoing data into the proper format.

If a protocol is chosen that does not match the choices shown for the protocol parameter, the following defaults are passed to the dlc hardware (parity is not set):

protocol	mode	datacode	parity	errorcheck
HDLC	SYNC_DCE	EBCDIC	----------	CRC_CCITT

Note

set_protocol() affects and is affected by the protocol analyzer's Setup Menu. It can change the system variables associated with its own parameters, the same variables that the Setup Menu uses. Any parameters successfully altered by the function appear in the Setup Menu even after program execution has terminated.

Similarly, if the NO_CHANGE choice is selected for any set_protocol() parameter, the function fetches the associated system variable, which may have been previously selected in the Setup Menu, and passes it to the dlc hardware.

Since the system variables are used in this way, set_protocol() does not have to be called in a program as long as its parameters have already been selected with the Setup Menu.

There are differences between set_protocol() and the Setup Menu. If the **Mode** and **Sync** softkeys are pressed in the Setup Menu, there is a choice of whether the clock source for the DTE channel is supplied from the DCE or the DTE device. In the set_protocol function, the same choice exists, but the mode and clock source are selected at the same time using the mode parameter. SYNC_DCE sets the mode to synchronous and selects the DCE device as the source for the DTE channel clock, while SYNC_DTE sets the mode to synchronous and selects the DTE device as the source for the DTE channel clock.

set_protocol( )

The datacode parameter affects how data is sent when the sendf() function is used. The ASCII string specified in the sendf() call is translated to the datacode specified by set_protocol(). The parity parameter affects sendf() in the same manner. While the datacode parameter does not affect the dlc, parity affects it when the COPS or BSC protocol is selected.

The value selected for errorcheck determines the error check used on all frames unless sendf() is called with BAD_FCS, ABORT_FCS or NO_FCS selected as the FCS parameter (see sendf(), conversion character %f).

Note If an unknown value is passed to one of the parameters, the default value for that parameter under the current protocol is used, and set_protocol() returns a warning (WARNING_1). If, however, the value passed is valid for the parameter, but incompatible with the current protocol and/or other parameters, the function returns in error (ERROR_1), and none of the requested changes are made.

Each protocol (starting with HDLC on the next page) is shown with its normal defaults ranged from left to right across the top row of choices. If a protocol is selected without defining the other parameters (NO_CHANGE is passed in instead), whatever values had previously been assigned to those parameters are used again.

For example, if the following function was called in a DataCommC program:

```
set_protocol(X25, SYNC_DCE, ASCII7, ODD_PARITY, CRC_CCITT);
```

the function would return successfully, and the protocol analyzer's Setup menu would reflect set_protocol()'s parameter values. Calling the function again with these values:

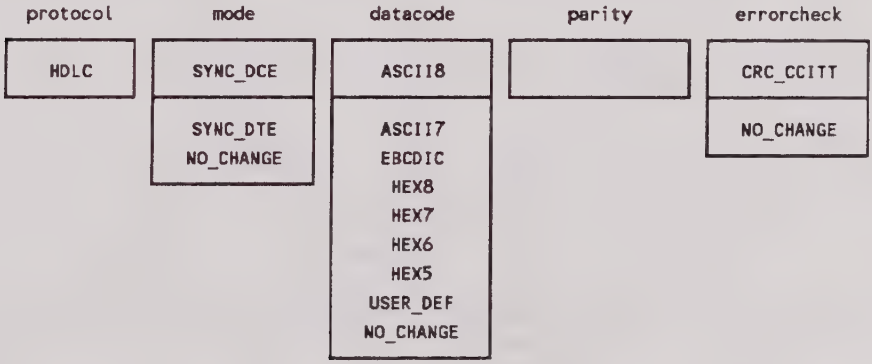
```
set_protocol(COPS, ASYNC_1, NO_CHANGE, NO_CHANGE, NO_CHANGE);
```

is equivalent to:

```
set_protocol(COPS, ASYNC_1, ASCII7, ODD_PARITY, CRC_CCITT);
```

which would return WARNING_1 since CRC_CCITT is not recognized by COPS. The default COPS errorcheck value, NO_ERROR_CHECK, would be substituted when the values were downloaded to the dlc.

HDLC, SDLC, X25 and X75 are all bit-oriented protocols (BOPS). Bit-oriented protocols use flag characters (0x7E) to idle the line; therefore the mode parameter choices include only the synchronous modes: SYNC_DCE and SYNC_DTE. SDLC also includes SYNC_NRZI as a mode choice.



HDLC, BSC and COPS protocols allow the choice of user-defined data codes; however, the data code cannot be defined in the DataCommC environment. The only way to define a data code is through the protocol analyzer's Setup Menu; the operation is described in appendix E of *The HP 4954A Operating Manual*. If USER_DEF is used as a datacode choice, whatever user-defined data code is specified in the Setup Menu is downloaded to the dlc. The Setup Menu has a default user-defined data code in which every character code is a "Don't Care" character; this data code is used when a data code has not been loaded or defined.

Although set_protocol() completely ignores whatever value is passed to parity for HDLC, a value must be passed in anyway.

set_protocol( )

protocol	mode	datacode	parity	errorcheck
SDLC	SYNC_DCE	EBCDIC		CRC_CCITT
	SYNC_DTE	ASCII8		NO_CHANGE
	SYNC_NRZI	HEX8		
	NO_CHANGE	NO_CHANGE		

Although `set_protocol()` completely ignores whatever value is passed in for `parity` in `SDLC`, a value must be passed in anyway.

protocol	mode	datacode	parity	errorcheck
X25	SYNC_DCE	ASCII8	NO_PARITY	CRC_CCITT
	SYNC_DTE	ASCII7	ODD_PARITY	NO_CHANGE
	NO_CHANGE	EBCDIC	EVEN_PARITY	
		HEX8	NO_CHANGE	
		NO_CHANGE		

protocol	mode	datacode	parity	errorcheck
X75	SYNC_DCE	ASCII8		CRC_CCITT
	SYNC_DTE	EBCDIC		NO_CHANGE
	NO_CHANGE	HEX8		
		NO_CHANGE		

Although `set_protocol()` completely ignores whatever value is passed in for `parity` in `X75`, a value must be passed in anyway.

set_protocol( )

The COPS protocol is used for character-oriented protocols. When COPS is specified, the set_sync, set_transtext, set_bcc and set_resync functions should be called to define other Setup Menu values. COPS also handles async protocols.

protocol	mode	datacode	parity	errorcheck
COPS	ASYNC_1	EBCDIC	NO_PARITY	NO_ERROR_CHECK
	ASYNC_1_5	ASCII8	ODD_PARITY	LRC
	ASYNC_2	ASCII7	EVEN_PARITY	CRC_6
	SYNC_DCE	HEX8	IGNORE	CRC_12
	SYNC_DTE	HEX7	NO_CHANGE	CRC_16
	NO_CHANGE	HEX6		NO_CHANGE
		HEX5		
		USER_DEF		
		NO_CHANGE		

BSC is a popular subset of COPS, but when it is used, the other Setup Menu values outlined in the description of COPS are automatically defined and the set_ functions do not need to be called.

protocol	mode	datacode	parity	errorcheck
BSC	SYNC_DCE	EBCDIC		CRC_16
	SYNC_DTE	ASCII7		LRC
	NO_CHANGE	USER_DEF		CRC_12
		NO_CHANGE		NO_CHANGE

Although set_protocol() completely ignores whatever value is passed in for parity in BSC, a value must be passed in anyway.

Return Values:

SUCCESSFUL (0)

set_protocol( )

WARNING_1 (1)	One or more parameter values are unknown. Default value(s) used. See text for explanations of the default values.
ERROR_1 (-1)	One or more parameter values are inconsistent. For example, this error would be returned if protocol==X25, datacode==ASCII8 and parity==ODD_PARITY, because this combination could not be created with the protocol analyzer's Setup Menu.
ERROR_10 (-10)	The data link control hardware has already been started. The dlc hardware must be stopped by calling stop_data() before any changes can be made to the dlc configuration.

See Also:

get_protocol(), get_mode(), get_datacode(), get_parity(), get_errorcheck(), set_sync(), set_transtext(), set_bcc(), set_resync()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int result;

   printf("Protocol parms set to X25, SYNC_DTE, ASCII8, NO_PARITY and CRC_CCITT");
   if((result = set_protocol(X25,SYNC_DTE,ASCII8,NO_PARITY,CRC_CCITT)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_protocol().\n",result);
      getch();
      exit();
   }
   getch();
}
```

set_resync( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_resync(rmode, numafter, itext, otext1, otext2, otext3, otext4, otext5, otext6)
int rmode, numafter;
char itext, otext1, otext2, otext3, otext4, otext5, otext6;
```

Description:

Defines the protocol rules for resynching, that is, dropping sync on the data stream and ignoring data on the line until the sync pattern is found again. This function should only be used with a character-oriented protocol (COPS in set_protocol(), or **Char Asyn/Syn** in the protocol analyzer's Setup Menu) in synchronous mode (SYNC_DCE or SYNC_DTE in set_protocol(), or **Sync** in the Setup Menu).

When the resync mode parameter, rmode, is set to MANUAL, the protocol analyzer does not drop sync until resync() is called, at which time sync is dropped immediately, without regard to any other factors. All other parameters (numafter, itext, otext1-6) are ignored when rmode is MANUAL. If rmode is set to AUTO, resynching is done automatically based on the numafter, itext, and otext1 - otext6 parameters.

The numafter parameter can be set to any integer between (and including) zero and 99. It defines how many characters to wait after the block check character (bcc) is received before dropping sync.

The itext parameter is an 8-bit value that tells the data link control hardware what character to drop sync on while in the transparent text mode. See set_transtext() for more information on the transparent text mode.

The otext parameters (otext1 - otext6) form an array of six 8-bit values defining characters that the data link control hardware drops sync on while outside of the transparent text mode.

set_resync( )

Return Values:

SUCCESSFUL (0)

WARNING_1 (1) The value of one or more of the parameter values is unknown. It is assumed to be the default. The mode parameter defaults to AUTO, numafter to 10, itext to 0x2d, and otext1, otext2, otext3, otext4, otext5 and otext6 to 0x2d, 0x37, 0x3d, 0x70, 0x7f and 0xff, respectively.

ERROR_1 (-1) The protocol is not COPS.

ERROR_2 (-2) The mode is not SYNC_DTE or SYNC_DCE.

ERROR_10 (-10) The protocol analyzer's data link control hardware was already running. The stop_data function must be called to halt the dlc hardware before any attempts to change the configuration.

See Also:

get_resyncafter(), get_resynchars(), get_resyncmode(), resync(), set_bcc(), set_protocol()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int result;

   if((result = set_protocol(COPS,SYNC_DTE,NO_CHANGE,NO_CHANGE,NO_CHANGE))
                                                        != SUCCESSFUL)
   {
      printf("Error #%d returned from set_protocol().\n", result);
      getch();
      exit();
   }
   if((result = set_resync(AUTO,10,0x2d,0x2d,0x37,0x3d,0x70,0x7f,0xff)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_resync().\n", result);
      getch();
      exit();
   }
   getch();
}
```

set_rows( )

Format:

```
void set_rows(min, max)
int min, max;
```

Description:

Sets the minimum and maximum rows to be written to using the screen output functions. This allows text to be confined to a certain area, or window, on the screen.

Return Values:

None

Example:

```
main()
{
   int i;

   set_rows(1, 10);
   for( i = 1; i < 20; i++)
      printf("This line will be displayed within rows 1 and 10.\n");
   getch();
}
```

set_screen_mode( )

Format:

```
#include <video.include>

void set_screen_mode(scrn_mode)
int scrn_mode;
```

Description:

Sets the screen mode to show a normal display (`scrn_mode = ASCII_DISPLAY`), or a display where whitespace and other transparent characters are represented by special characters (`scrn_mode = TRANSPARENT_DISPLAY`) instead of producing editing effects. For example, in `TRANSPARENT_DISPLAY`, a linefeed character would create a L_F character instead of advancing the cursor to the next row of text.

Return Values:

None

Example:

```
#include <video.include>

main()
{
   set_screen_mode(TRANSPARENT_DISPLAY);
   printf("The new line '\n' characters in this string will be displayed.\n");
   getch();
}
```

set_stop_states( )

Format:

```
#include <leads.include>
#include <retval.include>

int set_stop_states(lead_id, stop_state)
unsigned long lead_id;
int stop_state;
```

Description:

Sets the stop states of the specified interface lead (clocks and control leads). Only the leads that belong to the channel configured as the transmitter (see `set_channelconfig()`) may be specified in this function.

Return Values:

`SUCCESSFUL (0)`

`WARNING_1 (1)` There was an error in compatibility between the `lead_id` and the `stop_state` type. Trying to set a `stop_state` of `ON` for a clock lead, for instance, produces this return code. The following assumptions are made in this case:

All clock leads:	`ON` is replaced by `MARKING` or `ACTIVE`.
All clock leads:	`OFF` is replaced by `NOT_DRIVEN`.
All control leads:	`MARKING` or `ACTIVE` is replaced by `ON`.
All control leads:	`NOT_DRIVEN` is replaced by `OFF`.

Data lead stop states may not be modified.

`ERROR_1 (-1)` The `stop_state` parameter value was invalid. No stop states were set.

`ERROR_2 (-2)` The lead's channel was not configured (using `set_channelconfig()`) for transmit or for `SIMULATE`. No stop states were set.

ERROR_10 (-10) The protocol analyzer's data link control hardware was already running. The stop_data function should be called to halt the dlc hardware before any attempt to change the configuration.

See Also:

get_stop_states(), set_channelconfig()

Example:

```
#include <dlib.include>
#include <leads.include>
#include <retval.include>

main()
{
    int result;

    if((result = set_stop_states(RTS | DTR, ON)) != SUCCESSFUL)
    {
        printf("Error #%d returned from set_stop_states().\n",result);
        getch();
        exit();
    }
    getch();
}
```

set_sync( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_sync(sync_mode, synchar1, synchar2)
int sync_mode;
char synchar1, synchar2;
```

Description:

Sets the values of the sync characters and the synchronizaton mode. This function should only be used with a character-oriented protocol (COPS in set_protocol(), or **Char Asyn/Syn** in the protocol analyzer's Setup Menu) in synchronous mode (SYNC_DCE or SYNC_DTE in set_protocol(), or **Sync** in the Setup Menu).

The DataCommC event formatter uses sync character framing to create events from the data stream. When sync characters are not defined, the event formatter treats each character as an event, which significantly reduces system performance.

The sync_mode parameter may be set to ON or OFF (these constant values are located in the dlib.include file). If sync_mode is ON, the synchar1 and synchar2 parameters are used to define event framing. If sync_mode is OFF, the data are treated as if they are asynchronous, unframed characters, synchar1 and synchar2 are ignored, and every character forms an event.

The synchar1 and synchar2 parameters are 8-bit values that are passed to the data link control hardware when the start_data() is called. Datacode and parity (see set_protocol()) choices do not affect already-established synchar values.

Return Values:

SUCCESSFUL (0)

WARNING_1 (1)	The value of one or more of the parameter values is unknown. It is assumed to be the default. The sync_mode parameter defaults to ON, and both synchars default to 0x16.
ERROR_1 (-1)	The protocol is not COPS.
ERROR_2 (-2)	The mode is not SYNC_DTE or SYNC_DCE (see set_protocol()).
ERROR_10 (-10)	The protocol analyzer's data link control hardware was already running. The stop_data function should be called to halt the dlc hardware before any attempt to change the configuration.

See Also:

get_synchars(), set_resync(), set_protocol()

Example:

```
#include <dlib.include>
#include <retval.include>

main()
{
   int result;

   printf("Testing the set_sync function.");
   if((result = set_sync(ON, 0x32, 0x32)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_sync().\n", result);
      getch();
      exit();
   }
   getch();
}
```

set_time( )

Format:

```
#include <time.include>

void set_time(time)
struct tm *time;
```

Description:

Sets the current system time. When the address of an updated copy of the `tm` structure (pointed to by the `time` parameter) is passed to this function, the modified blocks in the structure are used to reprogram the real-time-clock. The `tm` data structure is defined in the `time.include` file (see chapter 4, "Include Files", for more information), and is outlined below.

struct tm

int	tm_msecs	0-900 milliseconds after the second (100 msec intervals)
int	tm_sec	0-59 seconds after the minute
int	tm_min	0-59 minutes after the hour
int	tm_hour	0-23 hours after midnight
int	tm_day	1-31 days of the month
int	tm_mon	1-12 months of the year
int	tm_year	0-99 years after turn of century

Return Values:

None

set_time( )

See Also:

get_time()

Example:

```
#include <time.include>

main()
{
    struct tm time;

    get_time(&time);
    printf("The date is %d/%d/%d and the old time is %d:%d:%d.%d.",
        time.tm_mon,
        time.tm_day,
        time.tm_year,
        time.tm_hour,
        time.tm_min,
        time.tm_sec,
        time.tm_msecs );

    time.tm_hour = 6;
    set_time(&time);
    get_time(&time);
    printf("The date is %d/%d/%d and the new time is %d:%d:%d.%d.",
        time.tm_mon,
        time.tm_day,
        time.tm_year,
        time.tm_hour,
        time.tm_min,
        time.tm_sec,
        time.tm_msecs );
    getch();
}
```

set_timer( )

Format:

```
int set_timer(number_of_tenths)
unsigned long number_of_tenths;
```

Description:

Sets one of the DataCommC timers for the duration, in tenths of seconds, specified in the `number_of_tenths` parameter.

Return Values:

Returns the number of the timer set (a non-negative integer), or

- `-1` if there is no memory for the timer, or if there are already 32,767 timers active. The request is ignored.
- `-2` if `number_of_tenths == 0`.

See Also:

`clear_timer(), wait()`

Example:

```
#include <retval.include>
#include <message.include>
#define FOREVER 1

main()
{
   int timer_number, result;
   long timer_duration = 10; /* set timeout for 1 second */
   MESSAGE msg;

   printf("Attempting to set the timer....\n");
   if((result = set_timer(timer_duration)) < 0)
   {
      printf("Error #%d returned from set_timer().\n",result);
      getch();
      exit();
   }

   timer_number = result;
   while(FOREVER)
   {
      if(read_message(&msg,1,0l))
         printf("Error reading message queue.\n");
      else
      {
         if((msg.type == TIMER) && (msg.subtype == TIMER_TIMEOUT))
            if(msg.body.timer_message.timer_number == timer_number)
            {
               printf("Timer %d has expired.\n", timer_number);
               break;
            }
      }
   }
   getch();
}
```

set_transtext( )

Format:

```
#include <dlib.include>
#include <retval.include>

int set_transtext(trans_mode, ttextchar)
int trans_mode;
char ttextchar;
```

Description:

Defines the protocol rules for transparent text mode when the protocol is COPS.

When the transparent text mode parameter, trans_mode, is OFF, transparent text characters are not recognized. If trans_mode is ON, the 8-bit value specified in ttextchar is used to signal a break into or out of the transparent text mode. While in transparent mode, all control characters are treated as data, that is, their defined control functions are not executed.

As mentioned above, the transparent text character, ttextchar, is an 8-bit value that defines the boundaries of the transparent text mode. It signals both the beginning and the end of the transparent text mode.

Return Values:

SUCCESSFUL (0)

WARNING_1 (1) The value of one or more of the parameter values is unknown. It is assumed to be the default. The trans_mode parameter defaults to OFF, in which case ttextchar is ignored. If trans_mode==ON, ttextchar may default to 0x10 (in ASCII, DLE - data link escape character).

ERROR_1 (-1) The protocol is not COPS.

ERROR_10 (-10) The protocol analyzer's data link control hardware was already running. The stop_data function must be called to halt the dlc hardware before any attempt to change the configuration.

See Also:

get_transtext()

Example:

```
#include <dlib.include>
#include <ascii.include>
#include <retval.include>

main()
{
   int result;

   if((result = set_transtext(ON,as_DLE)) != SUCCESSFUL)
   {
      printf("Error #%d returned from set_transtext().\n",result);
      getch();
      exit();
   }
   getch();
}
```

setjmp( )

Format:

```
#include <setjmp.include>

int setjmp(env)
jmp_buf env;
```

Description:

Saves the stack environment values in the memory block indicated by `env`, so that they may later be restored by `longjmp()`. The `longjmp` and `setjmp` functions are useful for dealing with errors encountered by the low-level functions of a program.

Return Values:

If `setjmp()` returns by way of a `longjmp()` call, it returns the `val` argument passed to `longjmp()`. Otherwise it returns zero.

See Also:

`longjmp()`

Example:

```
#include <setjmp.include>

main()
{
    jmp_buf buff;

    printf("We are about to call setjmp.\n");
    printf("Please hit any character to call it.\n");
    getch();
    if(setjmp(buff))
    {
        printf("We are in the if clause.\n");
        printf("We should have had an error and will return.\n");
    }
    else
    {
        printf("We are executing the else clause.\n");
        printf("We are now pretending to have an error.\n");
        printf("Hit any key to have a fake error and call longjmp.\n");
        getch();
        longjmp(buff,-1);
        printf("We have just called longjmp.  This should never get printed.\n");
    }
    printf("Please hit any key to return.\n");
    getch();
}
```

sin( )

Format:

```
#include <math.include>

double sin(x)
double x;
```

Description:

Calculates the sin of the angle x (where x is expressed in radians).

Return Values:

Returns the sine of x.

See Also:

asin(), atan(), atan2(), acos(), cos(), cosh(), cotan(), sinh(), tan(), tanh()

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double y, radx = PI/2;

   y = sin(radx);
   printf("The sine of PI/2 is %f.", y);
   getch();
}
```

sinh( )

Format:

```
#include <math.include>

double sinh(x)
double x;
```

Description:

Calculates the hyperbolic sin of x.

Return Values:

Returns the hyperbolic sine of x.

See Also:

```
asin(), atan(), atan2(), acos(), cos(), cosh(), cotan(), sin(), tan(), tanh()
```

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double y, radx = PI/2;

   y = sinh(radx);
   printf("The hyperbolic sine of PI/2 is %f.", y);
   getch();
}
```

spawn( )

Format:

```
#include <system.include>
#include <retval.include>

int spawn(path_name, number_of_user_args, user_args, priority, child_QID)
char *path_name;
int number_of_user_args;
char *user_args[];
int priority;
long *child_QID;
```

Description:

Creates a process that can execute independently of the calling process. The child process is spawned using system defaults and should be given a priority number between 1 and 199. If the priority is not within this range, `priority` is set to `DEFAULT_PRIORITY`, which is defined in the `system.include` file to be 100.

When the function returns successfully, the child's queue number is placed in the memory location pointed to by `child_QID`.

Null-terminated character strings can be passed to any spawned child process through the `number_of_user_args` and `user_args` arguments. The child process can gain access to these arguments through the `argc` and `argv` parameters of its own `main` function. Information from `number_of_user_args` is passed to `argc`, and information from `user_args` is passed to `argv`.

The `argc` parameter in `main()` is always greater than or equal to 1 because the run-time system inserts the child's path name into `argv[0]`. If the parent process passes a pointer to a string in `user_args[0]`, the child would use the element `argv[1]` to get a pointer to the string in the child's memory space.

Return Values:

`SUCCESSFUL (0)` A child process was spawned.

-1	The program prog_name was not found.
-2	There was no memory for the child process.
-3	Invalid program file format found.
-15 to -19	Can't communicate with ROOT process.
-21	No message queues were available.
-22	Can't notify process manager.
-41	Too many processes.
-42	No memory for user stack.
-43	User stack was too small.

See Also:

exit()

spawn( )

Example:

```
#include <retval.include>
#include <system.include>

main()
{
    int result, number_of_child_args = 2;
    char *argv[2];
    long child_qid;
    char input[81];
    char output[81];

    /* set up arguments to pass to child process */
    printf("Enter input file for child process:");
    gets(input);
    argv[0] = input;

    printf("Enter output file for child process:");
    gets(output);
    argv[1] = output;
    if((result = spawn("c:userprog.program",number_of_child_args,argv,
                            DEFAULT_PRIORITY, &child_qid)) != SUCCESSFUL)
    {
        printf("Error #%d returned from spawn().\n",result);
        getch();
        exit();
    }
    printf("The QID of the child_process is %lx.\n",child_qid);
    getch();
}
```

spawnw( )

Format:

```
#include <system.include>
#include <retval.include>

int spawnw(path_name, number_of_user_args, user_args, priority, child_exit_value)
char *path_name;
int number_of_user_args;
char *user_args[];
int priority;
int *child_exit_value;
```

Description:

Creates a process that can execute independently of the calling process. The child process is spawned using system defaults and should be given a priority number between 1 and 199. If the priority is not within this range, `priority` is set to `DEFAULT_PRIORITY`, defined to be 100.

If the child is successfully spawned, the calling process is blocked until the child process terminates using the `exit` function. After program execution is returned to the parent process, the `child_exit_value` parameter points to the integer value that the child specified when it called `exit()` (for example, `exit(0);`). If the child calls `exit()` without specifying an exit value, zero is returned to the parent through `child_exit_value`.

Null-terminated character strings can be passed to any spawned child process through the `number_of_user_args` and `user_args` arguments. The child process can gain access to these arguments through the `argc` and `argv` parameters of its own `main` function. Information from `number_of_user_args` is passed to `argc`, and information from `user_args` is passed to `argv`.

The `argc` parameter in `main()` is always greater than or equal to 1 because the run-time system inserts the child's path name into `argv[0]`. If the parent process passes a pointer to a string in `user_args[0]`, the child would use the element `argv[1]` to get a pointer to the string in the child's memory space.

spawnw( )

Return Values:

SUCCESSFUL (0)	A child process was spawned.
-1	The program prog_name was not found.
-2	There was no memory for the child process.
-3	Invalid program file format found.
-15 to -19	Can't communicate with ROOT process.
-21	No message queues were available.
-22	Can't notify process manager.
-41	Too many processes.
-42	No memory for user stack.
-43	User stack was too small.

See Also:

exit()

Example:

```
#include <retval.include>
#include <system.include>

main()
{
    int result, number_of_child_args = 2;
    char *argv[2];
    int child_exit_value;
    char input[81];
    char output[81];

    /* set up arguments to pass to child process */
    printf("Enter input file for child process:");
    gets(input);
    argv[0] = input;

    printf("Enter output file for child process:");
    gets(output);
    argv[1] = output;
    if((result = spawnw("c:userprog.program",number_of_child_args,argv,
                            DEFAULT_PRIORITY, &child_exit_value)) != SUCCESSFUL)
    {
        printf("Error #%d returned from spawn().\n",result);
        getch();
        exit();
    }
    printf("The child's exit value is %lx.\n",child_exit_value);
    getch();
}
```

sprintf( )

Format:

```
sprintf(str,format,args)
char *str;
char *format;
char *args;
```

Description:

Outputs or prints arguments in `args` to a memory location pointed to by `str` according to `format`.

The character string pointed at by `format` directs the output operation, and contains two types of information: ordinary alphanumeric characters, which are output unchanged; and conversion specifications, each of which causes the conversion and output of the next argument in the `args` list.

The formatted string is output from left to right. When a conversion specification is encountered, the next (initially first) argument is output according to the conversion specification.

A conversion specification has the form:

```
%[flag][width][.precision][l]type
```

Each field enclosed in braces "`[ ]`" is optional and consists of a single character or number signifying a particular format option.

`flag`

`-` The converted argument is left-justified when printed. The default is right-justification.

`width`

digit string The numeric digit string specifies the field width for the conversion. If the converted value has fewer characters than `width`, enough blank (space) characters are output to make the total number of characters output equal the field width. The spaces are output before or after the value, depending on the presence or absence of the left-justification flag. If the field width digits have a leading zero, zeros are used as pad characters instead of spaces. If the converted string has more characters than the value of `width`, the string is truncated.

`*` The `width` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`.precision`

digit string For floating point conversions, `precision` specifies the number of digits to appear after the decimal point; for character string conversions, it specifies the maximum number of characters to be printed from a string.

`*` The `precision` parameter is supplied by the corresponding argument in the argument list (`args`). The argument must be of type `int`.

`l`

A conversion normally performed on an `int` is performed on a `long` (may be used with the `d`, `o` and `x` conversion characters).

sprintf( )

The type character format is as follows:

character	type of argument	output format
d	int	signed decimal
u	int	unsigned decimal
x	int	unsigned hexadecimal
o	int	unsigned octal
f	float or double	floating point
c	char	single character
s	string	character string
e	float or double	scientific nototation
g	uses d, f or e - whichever gives	full precision in minimum space

Return Values:

Returns the number of characters in `str`, not including the terminating `NULL`.

See Also:

`printf(), fprintf()`

Example:

```
main()
{
   int j = 1;
   char outbuf[80];

   sprintf(outbuf,"The integer in hex is %x.\n",j);
   puts(outbuf);
   getch();
}
```

sqrt( )

Format:

```
#include <math.include>

double sqrt(x)
double x;
```

Description:

Calculates the square root of x.

Return Values:

Returns the result of the square root operation.

See Also:

```
exp(), log(), pow()
```

Example:

```
#include <math.include>

main()
{
   double x,y;

   x = 9;
   y = sqrt(x);
   printf("The square root of %3.1f = %3.1f.\n",x,y);
   getch();
}
```

srand( )

Format:

```
void srand(val)
int val;
```

Description:

Sets the seed of the pseudo-random number generator.

Return Values:

None

See Also:

```
rand()
```

Example:

```
main()
{
   unsigned i = 50;

   printf("Setting the random number generator seed to 50.");
   srand(i);
   getch();
}
```

sscanf( )

Format:

```
int sscanf(buffer, format, args)
char *buffer;
char *format;
char *args;
```

Description:

Takes text characters from the specified buffer or string (`buffer`), checks the character types against conversion characters imbedded in a control string pointed to by the `format` parameter, and places matching text in the fields pointed to by the `args` list.

The following `sscanf()` example shows the main components of the function:

```
sscanf(scanbuf, "%f%s", &fltptr, &strptr);
```

The string `"%f%s"` is a control string (`format`) with two control items (both conversion characters), one indicating a floating point value (`%f`), and the other (`%s`) indicating a string. The other arguments, `&fltptr` and `&strptr`, define the argument list (`args`). If `sscanf()` finds floating point characters, it places them in the memory location pointed to by `fltptr`; if a character string is found next, it is placed in the memory location pointed to by `strptr`.

A control string contains these control items:

- Conversion specifications
- Optional white space characters (tab, space, newline)
- Optional alphanumeric characters (not white space, and not part of a conversion specification).

The `sscanf` function works its way through a control string from left to right, trying to match each control item to a portion of the input stream. During the matching process, `sscanf()` fetches characters one at a time from the input stream. If a character is found which doesn't match the type specifier for the corresponding conversion specification, `sscanf()` pushes the character back onto the input stream and finishes processing the current control item. This

sscanf( )

"pushing back" frequently gives unexpected results when a stream is used later by other I/O functions, such as `getc()` or `scanf()`, as well as by `sscanf()` itself, if it is used again.

A conversion specification has the form:

```
%[*] [width] [l] type
```

Each field enclosed in braces "`[ ]`" is optional and consists of a single character or number signifying a particular format option. The simplest possible conversion specification contains a percent sign (`%`) and a conversion character (ex: `%f`)

The optional fields are defined below, and conversion characters are discussed later in this segment.

`*`

Assignment suppression character. The current stream is scanned, but not saved. The function goes on to the next control string item.

`width`

This field specifies the maximum number of characters to be fetched for the conversion.

`l`

This field indicates that the argument is a pointer to a long data type - the exact type (for example, long decimal, long hex, long unsigned) is determined by the conversion character.

The conversion character format is as follows:

character	type of argument	expected input format
d	pointer to int	signed decimal
u	pointer to int	unsigned decimal
x	pointer to int	unsigned hexadecimal
o	pointer to int	unsigned octal
e,f	pointer to float	floating point
c	pointer to char	single character
s	pointer to char array	character string

When a conversion specification is encountered in the control string, the sscanf function skips leading white space on the input stream, then collects characters from the stream until it encounters one that is not appropriate for the corresponding conversion character. That character is pushed back onto the input string.

As long as the conversion specification didn't request assignment suppression (see '*', above), the text string that was read from the keyboard is converted to the format specified by the conversion specification, the result is placed in the location pointed to by the corresponding args argument, the next argument becomes current, and the function proceeds to the next control string item.

If assignment suppression was requested, the sscanf function ignores the input characters and goes on the next control item.

If an ordinary character is found in the control string, outside any conversion specification, sscanf() fetches the next character. If that character matches the character in the control string, the function goes on to the next control string item, ignoring the input character. If there is no match, sscanf() terminates.

If a white space character is found in the control string, the sscanf function fetches input characters until the first non-white space character is read. The non-white space character is pushed back onto the input stream and sscanf() proceeds to the next item in the control string.

sscanf( )

Return Values:

Returns the number of items converted and assigned to memory locations in args. Unmatched items, since they are not assigned to args, are not included in the count.

See Also:

scanf(), fscanf()

Example:

```
main()
{
   int i;
   char *scanfbuf;

   scanfbuf = "100";
   sscanf(scanfbuf,"%d", &i);
   printf("The integer i = %d.\n",i);
   getch();
}
```

start_data( )

Format:

```
#include <retval.include>

int start_data()
```

Description:

Activates the protocol analyzer's data link control hardware and starts data aquisition. The source of data defaults to the protocol analyzer's setup value unless it has been changed with the `set_data_source` function. When `start_data()` runs, the contents of the data buffer are flushed, and events formed from the incoming data are put into the process queue. Leads, however, are not affected. Any of the leads that have been left in a stop state (with `set_stop_states()`) remain in that state after data aquisition ends.

If the source of data has been set to `DISC_RUN`, `set_data_source()` must be called before each `start_data()` call.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)`	There was an error in opening the data file.
`ERROR_2 (-2)`	The function encountered an unrecognized data link control data source (This error would only occur if the program's external data had been overwritten).
`ERROR_5 (-5)`	A fatal error occurred during an attempt to communicate with the dlc hardware.
`ERROR_9 (-9)`	There was no pod, or an unrecognized pod was attached to the protocol analyzer.
`ERROR_10 (-10)`	The dlc was already running.

start_data( )

ERROR_100 (-100) The protocol was not implemented.

See Also:

stop_data(), set_stop_states(), set_data_source()

Example:

```
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE msg;
   int result;

   if((result = start_data()) != SUCCESSFUL)
   {
      printf("Data Transmit/Receive cannot be started - error # %d.",result);
    - exit();
   }
   read_message(&msg, WAIT, WAIT_FOREVER);
   stop_data();
   printf("The message type read is: %d\n",msg.type);
   getch();
}
```

start_display( )

Format:

```
int start_display(file, screen_size, share_keyboard)
char *file;
int screen_size;
int share_keyboard;
```

Description:

Spawns a run-time procotol display. Only one display process may be active at a time. The display process is always given a priority (priority level 2) lower than any user process

The `file` parameter is the full path name of the desired protocol display program. The `c:C/Decode/BOPS.program` run-time display program is included with the DataCommC package. The display program may reside in any directory on the hard disc or on a flexible disc, but if a NULL character is passed to the file parameter, the default display is used, that is, the program that was chosen to be the default in the DataCommC Buffer Manager Menu.

A DataCommC program may share the screen with a display program by having the display program confine itself to a certain region of the screen. Although the display program's screen area always starts at the top of the screen, you can use the `screen_size` parameter to determine how many rows of the screen the display process may use. The `screen_size` parameter has a minumum value of 3 and a maximum value of 21 rows. The screen area cannot be less than 3 because the display process needs one row for its column heading, one for the line that shows the bottom of its screen area, and the third to display data. If `screen_size` is less than 3 or greater than 21, the display process uses the entire screen (21 rows plus the softkey area) and issues no warnings. The display process's softkey area is the bottom four rows of the screen.

If `share_keyboard` is `FALSE` (`0`), the display process generates a **Stop Run** softkey in the erightmost position (`SK8`), and when it is pressed, the display process exits (terminates itself) and notifies its parent process (your program) by sending it a message of type `DISPLAY` and subtype `STOPPED` (see `read_message()` and the include file `message.include`). Note that the display process does not stop the run; it is up to your program to do that when it receives the `DISPLAY/STOPPED` message.

start_display( )

In most cases, the display would be spawned with share_keyboard equal to FALSE, and a screen_size of 21 (the entire screen).

Normally your program would not be able to receive input from the keyboard while a protocol display process is running, since all keyboard input goes to that process. If, however, you start the display process with a share_keyboard parameter of TRUE (1), your program and the display process may pass control of the keyboard back and forth.

If share_keyboard is TRUE, the display process begins in its keyboard-off state, that is, it does not attempt to read input from the keyboard and refrains from writing to the softkey area of the screen. Your program begins with control of the keyboard and the softkey area. A "best practice" is to paint whatever softkeys are needed for your application, as well as one labeled **Display Control**, and then use a loop on getch() to handle keyboard inputs.

When your program detects that its **Display Control** softkey has been pressed, it should call keyboard_unlock() to give up control of the keyboard, and display_keys_on() to notify the display process that it now has control of the keyboard and softkey area. At this point, your program should refrain from calling getch() or writing to the softkey area, and should wait until the display process sends notification that it has relinquished control of the keyboard. This notification comes in the form of a message of type DISPLAY and subtype KEYBOARD_OFF which is generated when the display process's top level **Exit** softkey is pressed (when the keyboard is shared, the display process generates **Exit** instead of the **Stop Run** softkey).

Once this notification is received, your program should call keyboard_lock() to retake control of the keyboard, redisplay its softkeys, and continue looping on getch(). For an example of this type of program, see display_keys_on().

Return Values:

0	The display process was successfully spawned.
-1	The specified display program (file) was not found, or if file was NULL, the current default decode was not found.
-2	There wasn't enough memory for the display process.

-3	The specified program (file) was not a valid program.
-15 to -19	The function couldn't communicate with the ROOT process.
-21	There weren't any message queues available.
-22	The function couldn't notify the process manager.
-41	There were too many processes active.
-42	There wasn't any memory for the user stack.
-43	The user stack was too small.
-100	Another display process was already active.

See Also:

`stop_display()`

start_display( )

Example:

```
#include <stdio.include>
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE msg;
   int done, result;

   set_rows(17,21);
   result = start_display(NULL, 15, 0);
   printf("start_display() returned %d\n", result);
   done = 0;
   while(!done)
   {
      if(read_message(&msg, WAIT, WAIT_FOREVER) = SUCCESSFUL)
         if((msg.type == DISPLAY) && (msg.subtype == STOPPED))
         {
            done = 1;
            stop_display();
         }
   }
}
```

stop_data()

Format:

```
int stop_data()
```

Description:

Halts the protocol analyzer's data link control hardware and stops data aquisition and event formation. Leads are left in the state previously defined in `set_stop_states()`. If the data source had been set to `DISC_RUN` (see `set_data_source()`), the data file is closed.

Return Values:

`SUCCESSFUL (0)`

`ERROR_1 (-1)` The data link control hardware was not running when `stop_data()` was called.

See Also:

```
start_data()
```

stop_data( )

Example:

```
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE msg;
   int result;

   if((result = start_data()) != SUCCESSFUL)
      {
      printf("Data Transmit/Receive cannot be started - error # %d.", result);
      exit();
      }
   read_message(&msg, WAIT, WAIT_FOREVER);
   stop_data();
   printf("The message type read is: %d\n", msg.type);
   getch();
}
```

stop_display( )

Format:

```
int stop_display()
```

Description:

Terminates execution of the Run-time Display (decode) process.

Return Values:

0	The exit message was sent.
-1	The queue has been deleted
-2	An invalid queue number was given.
-3	Out of memory.
-5	Out of memory.
-6	The queue was full.

See Also:

```
start_display()
```

stop_display( )

Example:

```
#include <stdio.include>
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE msg;
   int done, result;

   set_rows(17,21);
   result = start_display(NULL, 15, 0);
   printf("start_display() returned %d\n", result);
   done = 0;
   while(!done)
   {
      if(read_message(&msg, WAIT, WAIT_FOREVER) = SUCCESSFUL)
         if((msg.type == DISPLAY) && (msg.subtype == STOPPED))
         {
            done = 1;
            stop_display();
         }
   }
}
```

strcat( )

Format:

```
#include <string.include>

strcat(s2,s1)
char *s2;
char *s1;
```

Description:

Copies, or concatenates, `s1` onto the end of `s2` (does not test for `s2` overflow).

Return Values:

None

See Also:

```
strncat()
```

Example:

```
#include <string.include>

main()
{
   char buf[30];

   strcpy(buf,"hello");
   strcat(buf," world");
   puts(buf);
   getch();
}
```

strchr( )

Format:

```
#include <string.include>

char *strchr(str,ch)
char *str;
char ch;
```

Description:

Returns a pointer to the first occurrence of character `ch` in string `str`, or `NULL` if not found (works even if `ch = NULL`). Use `strrchr()` to find the last occurrence of `ch` in `str`.

Return Values:

The return value is described above.

See Also:

```
strrchr()
```

Example:

```
#include <string.include>
main()
{
   char *buf, *substr;

   buf = "hello world";
   substr = strchr(buf,'w');
   printf("The remaining string is %s.\n",substr);
   getch();
}
```

strcmp( )

Format:

```
int strcmp(str1,str2)
char *str1;
char *str2;
```

Description:

Compares strings `str1` and `str2`. Returns a negative number if `str1 < str2`, zero if `str1 = str2`, and a positive number if `str1 > str2`. Uses a lexicographic comparison.

Return Values:

The return value is described above.

See Also:

`strcmpi(), strncmp()`

strcmp( )

Example:

```
main()
{
   char *str1, *str2;
   int result;

   str1 = "hello world";
   str2 = "hello World";
   if((result = strcmp(str1,str2)) < 0)
      printf("str1 < str2");
   else if (result == 0)
      printf("The two strings are equal");
   else
      printf("str1 > str2 ");
   getch();
}
```

strcmpi( )

Format:

```
int strcmpi(str1,str2)
char *str1;
char *str2;
```

Description:

Compares strings `str1` and `str2`. Returns a negative number if `str1 < str2`, zero if `str1 = str2`, and a positive number if `str1 > str2`. This routine is independent of case (upper or lower), and uses lexicographic comparison.

Return Values:

The return value is described above.

See Also:

`strcmp()`

strcmpi( )

Example:

```
main()
{
   char *str1, *str2;
   int result;

   str1 = "hello world";
   str2 = "hello World";
   if((result = strcmpi(str1,str2)) < 0)
      printf("str1 < str2");
   else if (result == 0)
      printf("The two strings are equal");
   else
      printf("str1 > str2 ");
   getch();
}
```

strcpy( )

Format:

```
#include <string.include>

char *strcpy(s2, s1)
char *s2;
char *s1;
```

Description:

Copies s1 to s2 up to and including the terminating NULL character.

Return Values:

Returns a pointer to s2.

See Also:

```
strncpy()
```

Example:

```
main()
{
   char buf[30];

   strcpy(buf,"hello world");
   puts(buf);
   getch();
}
```

strlen( )

Format:

```
int strlen(str)
char *str;
```

Description:

Finds the length of the string, `str` (strings are terminated by a `NULL` character).

Return Values:

Returns the string length.

Example:

```
#include <string.include>

main()
{
   char buf[30];
   int length;

   strcpy(buf,"hello world");
   length = strlen(buf);
   printf("The length of 'buf' is %d.\n", length);
   getch();
}
```

strncat( )

Format:

```
#include <string.include>

char *strncat(s2,s1,n)
char *s2;
char *s1;
int n;
```

Description:

Copies `n` characters (maximum) from `s1` onto the end of `s2`.

Return Values:

Returns a pointer to `s2`.

See Also:

`strcat()`

strncat( )

Example:

```
#include <string.include>

main()
{
   char buf1[30], *buf2;

   strcpy(buf1,"hello ");
   buf2 = "world";
   strncat(buf1,buf2,strlen(buf2));
   puts(buf1);
   getch();
}
```

strncmp( )

Format:

```
int strncmp(str1,str2,n)
char *str1;
char *str2;
int n;
```

Description:

Compares the first `n` characters of `str1` and `str2`. Returns a negative number if `str1 < str2`, zero if `str1 = str2`, and a positive number if `str1 > str2`. Uses a lexicographic comparison.

Return Values:

The return value is described above.

See Also:

`strcmp()`

strncmp( )

Example:

```
main()
{
    char *str1, *str2;
    int result;

    str1 = "hello world";
    str2 = "hello World";
    if((result = strncmp(str1,str2,5)) < 0)
       printf("str1 < str2");
    else if (result == 0)
       printf("The two strings are equal");
    else
       printf("str1 > str2 ");
    getch();
}
```

strncpy( )

Format:

```
#include <string.include>

char *strncpy(str2,str1,n)
char *str1;
char *str2;
int n;
```

Description:

Copies n characters from str1 to str2. If str1 contains fewer than n characters, str2 is padded with NULLs. If there are more than n characters in str1, only the first n characters are copied into str2, and the resulting string is not null-terminated.

Return Values:

Returns a pointer to str2.

See Also:

strcpy()

strncpy( )

Example:

```
#include <string.include>

main()
{
   char buf[30];

   strcpy(buf,"hello world",6);
   puts(buf);
   getch();
}
```

strrchr( )

Format:

```
#include <string.include>

char *strrchr(str,ch)
char *str;
char ch;
```

Description:

Returns a pointer to the last occurrence of character `ch` in string `str`, or `NULL` if not found (this works even if `ch = NULL`). Use `strchr()` to find the first occurrence of `ch` in `str`.

Return Values:

The return value is described above.

See Also:

`strchr()`

strrchr( )

Example:

```
#include <string.include>

main()
{
    char *buf, *substr;

    buf = "hello world";
    substr = strrchr(buf,'w');
    printf("The remaining string is %s.\n",substr);
    getch();
}
```

strsave( )

Format:

```
#include <string.include>

char *strsave(str)
char *str;
```

Description:

Allocates enough memory to hold the string pointed to by `str`, copies `str` into it, and returns a pointer to the new copy.

Return Values:

Returns a pointer to the new copy of the string, or `NULL` if there was no memory for `strsave()` to allocate.

Example:

```
#include <string.include>

main()
{
    char *buf;

    buf = strsave("hello world");
    puts(buf);
    getch();
}
```

strtrm( )

Format:

```
#include <string.include>

char *strtrm(string)
char *string;
```

Description:

Trims trailing spaces from a string.

Return Values:

Returns a pointer to string.

Example:

```
#include <string.include>

main()
{
   char str[30];

   strcpy(str,"hello world        ");
   strtrm(str);
   printf("The string equals %s.",str);
   getch();
}
```

sys_msg( )

Format:

```
void sys_msg(string)
char *string;
```

Description:

Writes the given string (`string`) to the screen (padded on either side with spaces to center it) with the current video attributes. This takes the whole row. Only the first 80 characters are written if the string is too long.

The cursor position is preserved.

Return Values:

None

Example:

```
main()
{

   sys_msg("This screen will be deleted when a key is pressed");
   getch();
}
```

tan( )

Format:

```
#include <math.include>

double tan(x)
double x;
```

Description:

Calculates the tangent of x (where x is expressed in radians).

Return Values:

Returns the tangent of x.

See Also:

```
asin(), atan(), atan2(), acos(), cos(), cosh(), cotan(), sin(), sinh(), tanh()
```

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double y, radx = PI/3;

   y = tan(radx);
   printf("The tangent of %f is %f.", radx, y);
   getch();
}
```

tanh( )

Format:

```
#include <math.include>

double tanh(x)
double x;
```

Description:

Calculates the hyperbolic tangent of x.

Return Values:

Returns the hyperbolic tangent of x.

See Also:

asin(), atan(), atan2(), acos(), cos(), cosh(), cotan(), sin(), sinh(), tan()

Example:

```
#include <math.include>
#define PI 3.141592654

main()
{
   double y, radx = PI/3;

   y = tanh(radx);
   printf("The hyperbolic tangent of %f is %f.", radx, y);
   getch();
}
```

toascii( )

Format:

```
#include <ctype.include>

int toascii(c)
char c;
```

Description:

Converts a single character, represented by the c parameter, to an ASCII character. The high-order bit of c is set to zero, converting the value to a valid member of the ASCII character set. If c was already an ASCII character, it remains unchanged.

Return Values:

Returns the character c.

See Also:

`isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace,(), isupper(), isxdigit(), tolower(), toupper()`

toascii( )

Example:

```
#include <ctype.include>

main()
{
   char ch;

   ch = 0x95;
   ch = toascii(ch);
   printf("The ascii character is %c",ch);
   getch();
}
```

tolower( )

Format:

```
#include <ctype.include>

int tolower(c)
char c;
```

Description:

Converts c to lowercase if c is uppercase.

Return Values:

Returns the lowercase version of the character c.

See Also:

```
isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(),
ispunct(), isspace,(), isupper(), isxdigit(), toascii(), toupper()
```

tolower( )

Example:

```
#include <ctype.include>
#include <stdio.include>

main()
{
   char ch;

   printf("Enter a character string for conversion - press RESET when done.\n");
   while(1)
   {
      ch = getc(stdin);
      if(isalpha(ch))
         switch(isupper(ch))
         {
            case 0:
               printf("The character converted to upper case is '%c'.\n",toupper(ch));
               break;
            default:
               printf("The character converted to lower case is '%c'.\n",tolower(ch));
               break;
         }
   }
}
```

toupper( )

Format:

```
#include <ctype.include>

int toupper(c)
char c;
```

Description:

Converts c to uppercase if c is an lowercase letter.

Return Values:

Returns the uppercase version of the character c.

See Also:

```
isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(),
ispunct(), isspace,(), isupper(), isxdigit(), toascii(), tolower()
```

Example:

```
#include <ctype.include>
#include <stdio.include>

main()
{
   char ch;

   printf("Enter a character string for conversion - press RESET when done.\n");
   while(1)
   {
      ch = getc(stdin);
      if(isalpha(ch))
         switch(isupper(ch))
         {
            case 0:
               printf("The character converted to upper case is '%c'.\n",toupper(ch));
               break;
            default:
               printf("The character converted to lower case is '%c'.\n",tolower(ch));
               break;
         }
   }
}
```

trigger_on_message( )

Format:

```
#include <message.include>

int trigger_on_message(message)
MESSAGE *message;
```

Description:

This routine reads (and blocks if necessary) messages from the process queue indicated by `message` and returns when one of the following conditions is met:

1) The message that was read has a message type and subtype that have been initialized as triggers in `init_trigger()`.

2) The message read has a subtype > 127 (negative). The `trigger_on_message` function returns on any subtype greater than 127. These subtypes are always initialized as triggers.

`DATACOMM` messages that do not cause `trigger_on_message()` to return are automatically released throught a call to `release_event()`.

Return Values:

Returns the type field of the message that caused `trigger_on_message()` to return.

See Also:

`init_trigger()`

Example:

```
#include <message.include>
#define The_sun_sets_in_the_west 1

main()
{
   MESSAGE message;
   int number_set;

   number_set = init_trigger(DATACOMM, 2 , DTE_FRAME, END_OF_DISC_RUN);
   printf("There are %d subtypes set by init_trigger for DATACOMM events.\n",
                                                                    number_set);
   number_set = init_trigger(TIMER, 32);
   printf("There are %d subtypes set by init_trigger for timer messages.\n",number_set);
   set_timer(10L);  /* timeout after 1 second */
   start_data(); /* start receiving DATACOMM events */

   while(The_sun_sets_in_the_west)
      switch(trigger_on_message(&message))
      {
         case DATACOMM:
            printf("Process DATACOMM messages here.\n");
            break;
         case TIMER:
            printf("Process TIMER messages here.\n");
            break;
         default: /* must be an error subtype */
            printf("Process ERROR messages here.\n");
      }
   getch();
}
```

ungetc( )

Format:

```
#include <stdio.include>

void ungetc(ch, file)
int ch;
FILE *file;
```

Description:

Pushes a character (`ch`) back onto an input stream. The stream may be a file, or the standard input device, `stdin`.

Return Values:

None

See Also:

`getc(), ungetch()`

ungetc( )

Example:

```
#include <stdio.include>

main()
{
    int ch = 0;

    printf("Enter a character from the keyboard:\n");
    ch = getc(stdin);
    ungetc(ch,stdin);
    putch(ch);
    getch();
}
```

ungetch( )

Format:

```
#include <conio.include>

int ungetch(ch)
int ch;
```

Description:

Places a keystroke (ch) back into the system queue to be read by getch().

Return Values:

SUCCESSFUL (0) There is no error return code.

See Also:

getch(), ungetc()

ungetch( )

Example:

```
#include <conio.include>
main()
{
    int ch = 0, result;
    char *str;

    ungetch(CLEAR_DISPLAY);
    str = "dlrow olleh";
    while(*str)
        if((result = ungetch(*str)) != 0)
            printf("Error ungetting %c - returned %x.\n",*str++,result);
        else
            str++;
    while ( ch != CLEAR_DISPLAY )
    {
        ch = getch();
        putch(ch);
    }
    getch();
}
```

unlink( )

Format:

```
int unlink (file)
char *file;
```

Description:

Deletes the file specified by `file`. The `file` parameter must include the path as well as the file name and extension for the file to be deleted.

Return Values:

`0`	The specified file was deleted.
`-1`	The file does not exist or could not be deleted.

See Also:

`open( )`

Example:

```
#include <stdio.include>
#include <fcntl.include>

main()
{
   int fdesc;
   long seek_result, write_result;
   char instr[20];

   if((fdesc = open("file1.text",O_WRONLY)) == -1 )
      printf("Can't open file1.text.");
   else
   {
      if(write_result = write(fdesc,"hello world\n",12) != 12 )
      {
         printf("Write error detected.");
         close(fdesc);
      }
      else
      {
         close(fdesc);
         if(unlink("file1.text"))
            printf("Error deleting the file");
         else
            printf("File has been deleted.\n");
      }
   }
   getch();
}
```

wait( )

Format:

```
int wait(number_of_tenths)
unsigned long number_of_tenths;
```

Description:

Blocks the calling process for the specified amount of time (measured in tenths of seconds).

Return Values:

SUCCESSFUL (0) There is no error return code.

See Also:

clear_timer(), set_timer()

Example:

```
main()
{
   printf("Press a key....\n");
   getch();
   printf("Going to wait for 5 seconds...\n");
   print_time();

   wait(50l);
   print_time();
   printf("Finished waiting...\n");
   getch();
}

#include "time.include"
print_time()
{
   struct tm time;

   get_time(&time);
   printf("The time is %d:%d:%d.%d\n",
          time.tm_hour,
          time.tm_min,
          time.tm_sec,
          time.tm_msecs );
}
```

wait_data( )

Format:

```
int wait_data(data_type, timeout, data_event)
int data_type;
long int timeout;
MESSAGE *data_event;
```

Description:

Gets the first message of type DATACOMM and subtype data_type in the message queue and places the message into the buffer pointed to by data_event. If no message is immediately available, the function waits timeout milliseconds for another one.

Return Values:

SUCCESSFUL (0)	A DATACOMM message was received.
WARNING_1 (1)	The message was not a DATACOMM message.
WARNING_2 (2)	The DATACOMM message's subtype was not the same as specified in data_type.
ERROR_1 (-1)	There wasn't any message in the queue.

Example:

```
#include <message.include>
#include <retval.include>

main()
{
   MESSAGE message;
   char *frame_type[2];
   FRAME_EVENT *frame_ptr;
   int result;
   frame_type[0] = "DTE frame";
   frame_type[1] = "DCE frame";

   if((result = start_data()) != SUCCESSFUL)
   {
      printf("Data Transmit/Receive cannot be started - error # %d.",result);
      exit();
   }
   while(1)
   {
      switch(result = wait_data(DTE_FRAME,50L,&message))
      {
         case SUCCESSFUL:
            frame_ptr = message.body.event.event_ptr;
            printf("The following frame has been received:\n");
            printf("The frame's fcs is: %d.\n",frame_ptr->fcs);
            printf("The frame's type is: %d = %s.\n",
                  message.subtype,frame_type[message.subtype-1]);
            printf("The frame's length is: %d.\n",frame_ptr->length);
            printf("The frame's data starts at address: %lx.\n",frame_ptr->frame);
            release_event(message.body.event.event_ptr);
            break;
         case WARNING_1:
            printf("The message was not a a Datacomm message.\n");
            release_event(message.body.event.event_ptr);
            break;
```

wait_data( )

```
            case WARNING_2:
                printf("The datacomm message was not a frame event.\n");
                release_event(message.body.event.event_ptr);
                break;
            case ERROR_1:
                printf("There is no message in the datacomm queue.\n");
                break;
            default:
                printf("Error code returned from wait_data is %d.\n",result);
                break;
        }
        getch();
    }
}
```

wait_lead()

Format:

```
#include <leads.include>

int wait_lead(lead_id, lead_state, timeout, lead_event);
long lead_id;
int lead_state;
unsigned long timeout;
MESSAGE *lead_event;
```

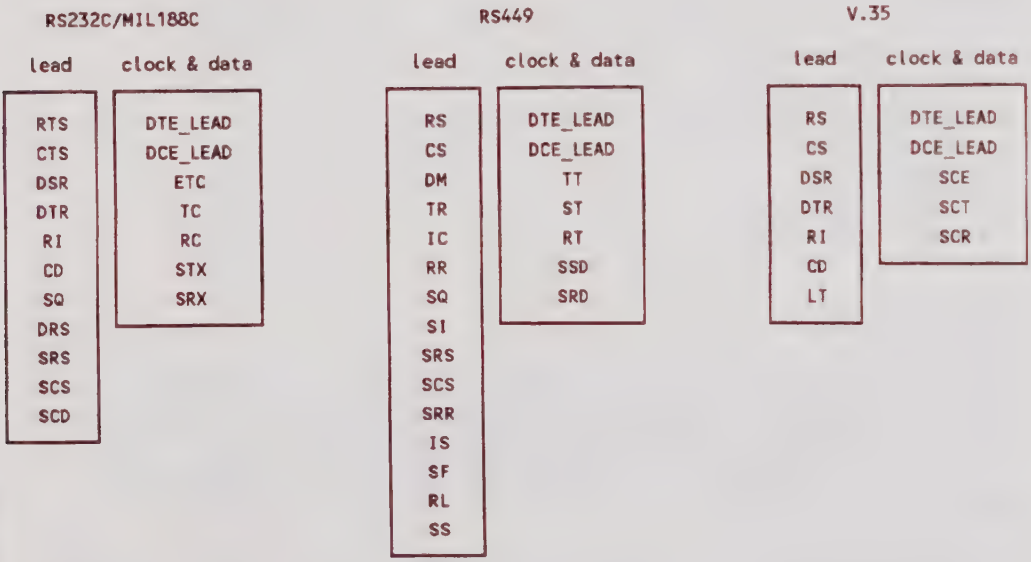
Description:

Scans the message queue and collects the first message of type DATACOMM and subtype LEAD_CHANGE with lead_id and lead_state as specified in the parameter list. When the desired message is found, the message is copied to the buffer pointed to by lead_event. If such a message cannot not found, the function waits for the duration of timeout (measured in milliseconds) before exiting.

DATACOMM messages that do not contain the correct lead id and state are automatically released through release_event().

Like set_lead() and get_lead(), wait_lead() uses the file leads.include to define valid lead names and states. Recognized lead states are ON, OFF, and DONT_CARE; Valid lead names are shown on the next page:

wait_lead( )



Return Values:

SUCCESSFUL (0)

ERROR_2 (-2) The lead_state parameter was invalid.

ERROR_5 (-5) The timeout occurred before desired event was found. The lead_event parameter is set to NULL.

ERROR_6 (-6) System timers could not be set.

See Also:

get_lead(), set_lead()

Example:

```
#include <message.include>
#include <leads.include>
#include <dlib.include>
#include <retval.include>

main()
{
   MESSAGE message;
   if(start_data())
   {
      printf("Data Transmit/Receive cannot be started - error # %d.",result);
      exit();
   }
   while(1)
   {
      switch(wait_lead(DTR,OFF,50L,&message))
      {
         case SUCCESSFUL:
            printf("DTR has been set OFF.\n");
            release_event(message.body.event.event_ptr);
            break;
         case ERROR_2:
            printf("Invalid lead_state parameter.\n");
            break;
         case ERROR_5:
            printf("Timeout occurred before the desired event was found.\n");
            break;
         case ERROR_6:
            printf("Unable to set the system timers.\n");
            break;
      }
   getch();
   }
}
```

wipe( )

Format:

```
void wipe(row1, row2)
int row1, row2;
```

Description:

Clears the screen from row1 to row2.

Return Values:

None

See Also:

```
write_vidram()
```

Example:

```
main()
{
   int i;

   set_rows(5, 15);
   for(i = 1; i < 30; i++)
      printf("This is row %d\n", i);
   printf("Press any key to clear the screen");
   getch();
   wipe(5,20);
   printf("Press any key to exit");
   getch();
}
```

write( )

Format:

```
#include <stdio.include>

int write(fdesc, buff, cnt)
int fdesc;
char buff[];
unsigned int cnt;
```

Description:

Writes `cnt` bytes from the storage buffer `buff[]`, to the file whose descriptor is `fdesc`. The `fdesc` parameter contains an integer that was generated by the `open` function when the file was first encountered in the program. The number of bytes transferred cannot exceed 64K.

Return Values:

Returns the number of bytes actually written, which may be less than `cnt` if the function runs out of disc or file space before `cnt` bytes are written.

See Also:

`open(), read()`

write( )

Example:

```
#include <stdio.include>
#include <fcntl.include>

main()
{
    int fdesc;
    long seek_result, write_result;
    char instr[20];

    printf("The string \"hello world\" will be written to file1.text.\n");
    if((fdesc = open("file1.text",O_WRONLY|O_CREAT)) == -1)
       printf("Can't create file1.text.");
    else
    {
        if(write_result = write(fdesc,"hello world\n",12) != 12)
           printf("Write error detected.");
        close(fdesc);
    }
    getch();
}
```

write_rs232( )

Format:

```
void write_rs232(ch)
char ch;
```

Description:

Writes the character ch to the RS232 port.

This function must not be used until `init_rs232()` has been called.

Return Values:

None

See Also:

`init_rs232()`, `rs232_read_ready()`, `rs232_write_ready()`, `print_char()`, `read_rs232()`

write_rs232( )

Example:

```
#include <conio.include>

/* A Dumb terminal emulator, terminates after the '~' character is typed */
/* on the keyboard....                                                    */

main()
{
   char ch = 0;

   init_rs232();

   printf("Terminal Emulator - press \"~\" to exit.");
   while(ch != '~' )
   {
      if(is_key_avail())
      {
         ch = getch();
         while (!rs232_write_ready()) ;
            write_rs232(ch);
      }
      if(rs232_read_ready())
      {
         read_rs232(&ch);
         putch(ch);
         ch = 0;
      }
   }
}
```

write_vidram( )

Format:

```
#include <video.include>

void write_vidram(row, column, character, attribute)
int row, column;
int attribute;
char character;
```

Description:

Writes `character` and its `attribute` to the video RAM location indicated by `row` and `column`. The formatted character appears on the display in the position `row, column`. This function is not affected by the text window areas created by set_rows(), and allows text and messages to be printed to any area of the display.

This function has no effect on the current cursor position.

Return Values:

None

See Also:

```
read_vidram()
```

write_vidram( )

Example:

```
#include <video.include>

main()
{
    char ch;
    int row = 1, col = 1;
    int attribute;
    char *outstr;

    outstr = "123456789012345678901234 5";
    for(;row <= 25;row++,col++)
        if(row & 0x0001)
            write_vidram(row,col,*outstr++,NORMAL);
        else
            write_vidram(row,col,*outstr++,BLINK);
    getch();
}
```

Include Files

The files listed in this chapter are located in the `c:C/Include` directory. Include files are stored as text, so they can be edited or printed through the Development Environment. Figure 4-1 shows a list of the include files in this chapter:

ascii.include	message.include
assert.include	retval.include
conio.include	setjmp.include
ctype.include	stdio.include
decode.include	stdlib.include
dlib.include	string.include
ebcdic.include	system.include
fcntl.include	time.include
leads.include	video.include
math.include	

Figure 4-1. Include files in the c:C/Include directory

When used in source code, the include file names must be entered as lowercase letters, but the `.include` extension may be replaced by `.h`; for example, the line "`#include leads.include`" may be represented by "`#include leads.h`" in a `.csource` file.

ascii.include

This include file contains ASCII values for data codes.

```
#define as_NUL          0
#define as_SOH          1
#define as_STX          2
#define as_ETX          3
#define as_EOT          4
#define as_ENQ          5
#define as_ACK          6
#define as_BEL          7
#define as_BS           8
#define as_HT           9
#define as_LF           10
#define as_VT           11
#define as_FF           12
#define as_CR           13
#define as_SO           14
#define as_SI           15
#define as_DLE          16
#define as_DC1          17
#define as_DC2          18
#define as_DC3          19
#define as_DC4          20
#define as_NAK          21
#define as_SYN          22
#define as_ETB          23
#define as_CAN          24
#define as_EM           25
#define as_SUB          26
#define as_ESC          27
#define as_FS           28
#define as_GS           29
#define as_RS           30
#define as_US           31
```

assert.include

This include file contains the assert macro.

```
#ifndef NDEBUG
#ifndef stderr
#include <stdio.include>
#endif
#define assert(x) if (!(x)) (fprintf(stderr,"Assertion failed: x, file %s, line
%d\n",__FILE__,__LINE__); error_notification("Assertion Failed: ",0);)
#else
#define assert(x)
#endif
```

conio.include

This include file gives definitions for console I/O codes and keyboard values.

```
#define LINEFEED                0x00A
#define RETURN                  0x00D
#define CARRIAGE                RETURN

/* Key definitions */
#define CLEAR_LINE              0x109
#define CLEAR_DISPLAY           0x10A
#define HOME                    0x10B
#define SK1                     0x101
#define SK2                     0x102
#define SK3                     0x103
#define SK4                     0x104
#define SK5                     0x105
#define SK6                     0x106
#define SK7                     0x107
#define SK8                     0x108
#define INSERT_CHAR             0x110
#define DELETE_CHAR             0x111
#define ROLL_UP                 0x112
```

```
#define UP_ARROW                0x113
#define NEXT_PAGE               0x114
#define LEFT_ARROW              0x115
#define BACKSPACE               0x115
#define RIGHT_ARROW             0x116
#define ROLL_DOWN               0x117
#define DOWN_ARROW              0x118
#define PREV_PAGE               0x119

/* shift - key values */
#define SHFT_CLEAR_LINE         0x309
#define SHFT_CLEAR_DISPLAY      0x30A
#define SHFT_HOME               0x30B
#define SHFT_SK1                0x301
#define SHFT_SK2                0x302
#define SHFT_SK3                0x303
#define SHFT_SK4                0x304
#define SHFT_SK5                0x305
#define SHFT_SK6                0x306
#define SHFT_SK7                0x307
#define SHFT_SK8                0x308
#define SHFT_RETURN             0x30D
#define SHFT_INSERT_CHAR        0x310
#define SHFT_DELETE_CHAR        0x311
#define SHFT_ROLL_UP            0x312
#define SHFT_UP_ARROW           0x313
#define SHFT_NEXT_PAGE          0x314
#define SHFT_LEFT_ARROW         0x315
#define SHFT_RIGHT_ARROW        0x316
#define SHFT_ROLL_DOWN          0x317
#define SHFT_DOWN_ARROW         0x318
#define SHFT_PREV_PAGE          0x319

/* control-key values */
#define CTRL_CLEAR_LINE         0x509
#define CTRL_CLEAR_DISPLAY      0x50A
#define CTRL_HOME               0x50B
#define CTRL_SK1                0x501
#define CTRL_SK2                0x502
#define CTRL_SK3                0x503
#define CTRL_SK4                0x504
#define CTRL_SK5                0x505
```

```
#define CTRL_SK6                0x506
#define CTRL_SK7                0x507
#define CTRL_SK8                0x508
#define CTRL_RETURN             0x50D
#define CTRL_INSERT_CHAR        0x510
#define CTRL_DELETE_CHAR        0x511
#define CTRL_ROLL_UP            0x512
#define CTRL_UP_ARROW           0x513
#define CTRL_NEXT_PAGE          0x514
#define CTRL_LEFT_ARROW         0x515
#define CTRL_RIGHT_ARROW        0x516
#define CTRL_ROLL_DOWN          0x517
#define CTRL_DOWN_ARROW         0x518
#define CTRL_PREV_PAGE          0x519
```

ctype.include

This include file defines the character type and character conversion macros.

```
extern char ctp_[];

#define isalpha(x)   (ctp_[(x)+1]&0x03)
#define isupper(x)   (ctp_[(x)+1]&0x01)
#define islower(x)   (ctp_[(x)+1]&0x02)
#define isdigit(x)   (ctp_[(x)+1]&0x04)
#define isxdigit(x)  (ctp_[(x)+1]&0x08)
#define isalnum(x)   (ctp_[(x)+1]&0x07)
#define isspace(x)   (ctp_[(x)+1]&0x10)
#define ispunct(x)   (ctp_[(x)+1]&0x40)
#define iscntrl(x)   (ctp_[(x)+1]&0x20)
#define isprint(x)   (ctp_[(x)+1]&0xc7)
#define isgraph(x)   (ctp_[(x)+1]&0x47)
#define isascii(x)   (((x)&0x80)==0)

#define toascii(x)          ((x)&127)
#define _tolower(x)         ((x)|0x20)
#define _toupper(x)         ((x)&0x5f)
```

decode.include

This include file contains definitions and data structures necessary for writing decodes.

```
/* values for key[].style */
#define BLANK 0   /* softkey is blank and hitting it does nothing */
#define OFFON 1   /* softkey is a toggle (half-bright or full) */
#define CYCLE 2   /* stay on same level & cycle through the labels */
#define ZCYCL 3   /* same as CYCLE, but values start at 0 */
#define LAYER 4   /* create a lower level of softkeys */

typedef struct decode_keys
{
   int style;
   char *label[8];
   int value;
   struct decode_keys *next_layer;
}DECODE_KEYS;
```

dlib.include

This include file contains definitions for datacomm library functions.

```
#define ON                    0x1
#define OFF                   0x0

#define YES                   0x1
#define NO                    0x0

#define ALL                   0x0

/* set_data_source() arguments */
#define LINE_RUN              0x2
#define DISC_RUN              0x3

/* FCS  TYPES */
#define GOOD_FCS              0x0
```

```
#define BAD_FCS                 0x1
#define ABORT_FCS               0x2
#define NO_FCS                  0x3

/* CHANNEL CONFIGURATIONS - used by set_channelconfig() */
#define NONE                    0x0
#define RX                      0x1
#define TX                      0x2

/* POD TYPES - used by read_pod_id() */
#define RS232                   0x1
#define V_35                    0x2
#define RS449                   0x3
#define X_21                    0x4
#define MIL188C                 0x5
#define NO_POD                  0x8

/* PROTOCOLS - used by set_protocol() */
#define BSC                     0x1
#define SDLC                    0x2
#define HDLC                    0x3
#define X25                     0x4
#define COPS                    0x5
#define X75                     0x7

/* SYNCHRONIZATION MODES */
#define ASYNC_1                 0x1
#define ASYNC_1_5               0x2
#define ASYNC_2                 0x3
#define SYNC_DCE                0x4
#define SYNC_NRZI               0x5
#define SYNC_DTE                0x6

/* DATACODES */
#define ASCII8                  0x0001
#define ASCII7                  0x0002
#define EBCDIC                  0x0003
#define HEX8                    0x0004
#define HEX7                    0x0005
#define HEX6                    0x0006
#define HEX5                    0x0007
#define USER_DEF                0x0008
```

```
#define NO_CHANGE               0xffff

/* PARITY */
#define ODD_PARITY              0x1
#define EVEN_PARITY             0x2
#define NO_PARITY               0x3
#define IGNORE                  0x4

/* COPS error type values */
#define GOOD_BCC                0x04
#define BAD_BCC                 0x05
#define PARITY_ERROR            0x06

/* ERRORCHECKS */
#define CRC_16                  0x1
#define CRC_CCITT               0x2
#define NO_ERROR_CHECK          0x3
#define CRC_12                  0x4
#define LRC                     0x5
#define CRC_6                   0x6

/* COPS/ASYNC */
#define MANUAL                  0x1
#define AUTO                    0x2

/* CHANNELS */
#define DCE                     0x1
#define DTE                     0x2
#define PRIM_ALL                0x3
#define SDTE                    0x4
#define SDCE                    0x5

/* duplex types used in set_duplex() */
#define HDX                      1
#define FDX                      2

/* lead control constants used in set_lead_control() */
#define LEAD_CONTROL_DEFAULT     2
#define LEAD_CONTROL_USER_DEF    1
```

ebcdic.include

This include file contains EBCDIC values for data codes.

```
#define eb_NUL          0
#define eb_SOH          1
#define eb_STX          2
#define eb_ETX          3
#define eb_EOT         55
#define eb_ENQ         45
#define eb_ACK         46
#define eb_BEL         47
#define eb_BS          22
#define eb_HT           5
#define eb_LF          37
#define eb_VT          11
#define eb_FF          12
#define eb_CR          13
#define eb_SO          14
#define eb_SI          15
#define eb_DLE         16
#define eb_DC1         17
#define eb_DC2         18
#define eb_DC3         19
#define eb_DC4         60
#define eb_NAK         61
#define eb_SYN         50
#define eb_ETB         38
#define eb_CAN         24
#define eb_EM          25
#define eb_SUB         63
#define eb_ESC         39
#define eb_FS          28
#define eb_GS          29
#define eb_RS          30
#define eb_US          31
```

fcntl.include

This include file gives definitions for file control parameters.

```
#define O_RDONLY            0
#define O_WRONLY            1
#define O_RDWR              2
#define O_CREAT          0x0100
#define O_TRUNC          0x0200
#define O_EXCL           0x0400
#define O_APPEND         0x0800
```

leads.include

This include file gives definitions for datacomm lead mapping.

```
/*  RS232 and MIL188C lead name to lead_id mapping  */
#define RTS                 0x0001l
#define CTS                 0x0002l
#define DSR                 0x0004l
#define DTR                 0x0008l
#define RI                  0x0010l
#define CD                  0x0020l
#define SQ                  0x0040l
#define DRS                 0x0080l
#define SRS                 0x0100l
#define SCS                 0x0200l
#define SCD                 0x0400l

/* RS232 and MIL188C clock and data name  mappings */
#define DTE_LEAD            0x010000l
#define DCE_LEAD            0x020000l
#define ETC                 0x040000l
#define TC                  0x080000l
#define RC                  0x0100000l
#define STX                 0x0200000l
#define SRX                 0x0400000l
```

```
/*  RS449 lead name to lead_id mapping  */
#define RS                      0x0001l
#define CS                      0x0002l
#define DM                      0x0004l
#define TR                      0x0008l
#define IC                      0x0010l
#define RR                      0x0020l
/*      SQ                      0x0040l  see RS232 definition */
#define SI                      0x0080l
/*      SRS                     0x0100l  see RS232 definition */
/*      SCS                     0x0200l  see RS232 definition */
#define SRR                     0x0400l
#define IS                      0x0800l
#define SF                      0x01000l
#define RL                      0x02000l
#define SS                      0x04000l

/* RS449 clock and data name  mappings */
/*      DTE_LEAD                0x010000l  see RS232 definition */
/*      DCE_LEAD                0x020000l  see RS232 definition */
#define TT                      0x040000l
#define ST                      0x080000l
#define RT                      0x0100000l
#define SSD                     0x0200000l
#define SRD                     0x0400000l

/*  V_35 lead name to lead_id mapping  */
/*      RS                      0x0001l  see RS449 definition */
/*      CS                      0x0002l  see RS449 definition */
/*      DSR                     0x0004l  see RS232 definition */
/*      DTR                     0x0008l  see RS232 definition */
/*      RI                      0x0010l  see RS232 definition */
/*      CD                      0x0020l  see RS232 definition */
#define LT                      0x0040l

/* V_35 clock and data name  mappings */
/*      DTE_LEAD                0x010000l  see RS232 definition */
/*      DCE_LEAD                0x020000l  see RS232 definition */
#define SCE                     0x040000l
#define SCT                     0x080000l
#define SCR                     0x0100000l
```

```
/* Lead state mappings */
#define NOT_DRIVEN              0x02
#define DONT_CARE               0x0FF

/* Clock and data state mappings */
#define MARKING                 0x03
#define ACTIVE                  0x04

/* RS232 ignored lead definitions */
#define IGNORE_NO_LEADS         0x00000
#define IGNORE_ALL_LEADS        0x0FFFF
#define DEFAULT_IGNORED_LEADS   0x0F7D0  /* ignore: RI,SQ,DRS,SRS,SCS,SCD */

/* RS232 stop state definitions  */
#define  DEFAULT_STOP_STATES    0x00

/* ISDN lead mapping */
#define INFO_0_TE               0x1
#define INFO_0_NT               0x2
#define INFO_2_NT               0x4
#define INFO_1_TE               0x8
#define RESYNC_NT               0x10
#define INFO_4_NT               0x20
#define SOURCE_1_ON             0x80
#define INFO_3_TE               0x100
#define SOURCE_1_NORM           0x200
#define SOURCE_2_ON             0x400
#define RESYNC_TE               0x800
#define CHAN_B                  0x4000
```

math.include

This include file defines math functions and boundary variables.

```
double sin(), cos(), tan(), cotan();
double asin(), acos(), atan(), atan2();
double ldexp(), frexp(), modf();
double floor(), ceil(), fabs();
double log(), log10(), exp(), sqrt(), pow();
double sinh(), cosh(), tanh();

#define HUGE_VAL          1.79e+308
#define LOGHUGE             709.778
#define TINY_VAL           2.2e-308
#define LOGTINY            -708.396
```

message.include

This include file defines parameters and data structures used to send and receive messages.

```
#define INT8          unsigned char
#define INT16         unsigned int
#define INT32         unsigned long

/*******************************/
/* send_message() modes        */
/*******************************/
#define ZERO                     0
#define EXPEDITE                 1

/*******************************/
/* send_message() error codes */
/*******************************/
#define QUEUE_DELETED           -1      /* queue no longer available */
#define INVALID_QUEUE_NUMBER    -2      /* queue number has not been used */
#define NO_QUEUE_BUFFERS        -5      /* maximum # of message buffers reached */
#define MAX_MSGS_EXCEEDED       -6      /* queue is full */
```

```
/*******************************/
/* read_message() error codes */
/*******************************/
#define NO_MESSAGE              -1
/* the following send_message() error code is also returned by read_message():   */
/*      INVALID_QUEUE_NUMBER                                                     */

/*************************************/
/* read_message() wait constants   */
/*************************************/
#define WAIT                     1
#define NO_WAIT                  0

/*************************************/
/* read_message() timeout constant */
/*************************************/
#define WAIT_FOREVER             0L

/*************************************/
/* DataCommC MESSAGE TYPES          */
/*************************************/

/*************************************/
/* MESSAGE TYPES 0-199 reserved     */
/* for HP internal use              */
/*************************************/
#define ERROR                   200
#define DATACOMM                201
#define TIMER                   202
#define DISPLAY                 203
```

```
/*************************************/
/*************************************/
/*       USER MESSAGE TYPES          */
/* These types are compatible with   */
/* trigger_on_message()              */
/*************************************/
/*************************************/
#define USER_MESSAGE_TYPE_1      229
#define USER_MESSAGE_TYPE_2      230
#define USER_MESSAGE_TYPE_3      231

/*****************************************/
/* The message types 232-255 are         */
/* reserved for the user but are not     */
/* compatible with trigger_on_message() */
/*****************************************/
#define USER_TYPE_1              232
#define USER_TYPE_2              233
/* the values 232 - 255 can be defined by the user */

/*************************************/
/*************************************/
/* MESSAGE SUBTYPES                  */
/*************************************/
/*************************************/

/*************************************/
/* DATACOMM subtypes                 */
/*************************************/
#define DTE_FRAME                 1
#define DCE_FRAME                 2
#define SDTE_FRAME                3
#define SDCE_FRAME                4
#define LEAD_CHANGE               5
#define DTE_CHAR                  6
#define DCE_CHAR                  7
#define DTE_ERCK                  8
#define DCE_ERCK                  9
#define END_OF_DISC_RUN          31
```

```
/*************************************/
/* DATACOMM Error subtypes           */
/*************************************/
#define DTE_OVERRUN              -1
#define DCE_OVERRUN              -2
#define SDTE_OVERRUN             -3
#define SDCE_OVERRUN             -4
#define EVENT_OVERRUN          -100
#define QUEUE_ERROR            -101

/*************************************/
/* Timer subtypes                    */
/*************************************/
#define TIMER_TIMEOUT             3

/*************************************/
/* Display subtypes                  */
/*************************************/
#define KEYBOARD_OFF              1
#define STOPPED                   2

/*********************************************************************/
/* Error subtypes (and error values used by error_notification()) */
/*********************************************************************/
#define FATAL_ERROR               1
#define NON_FATAL_ERROR           0

/*****************************************************************/
/* Top and Bottom boundries for the 3000 element event array    */
/*****************************************************************/
#define TOP_EVENT    ((void *) 0x0D20100) /* start of 3000 element EBS       */
                                          /* event array.                    */

#define BOTTOM_EVENT ((void *) 0x0D2BC70) /* start of last element of event  */
                                          /* array                           */
```

```
/*********************/
/* Character Event:  */
/*********************/
typedef struct
{
   INT8  subtype;      /* DTE_CHAR, DCE_CHAR */
   INT8  held_count;   /* how many processes are holding this event        */
   INT16 lead_status;  /* bit-mapped status of up to 45 leads              */
   INT16 msec;         /* 0-999 */
   INT8  second;       /* 0-59 */
   INT8  minute;       /* 0-59 */
   INT8  hour;         /* 0-23 */
   INT8  data_char;    /* character data */
} CHAR_EVENT;

/* macros to access character event fields */
#define CHE_subtype(msg)     ((CHAR_EVENT *)(msg.body.event.event_ptr))->subtype
#define CHE_held_count(msg)  ((CHAR_EVENT *)(msg.body.event.event_ptr))->held_count
#define CHE_lead_status(msg) ((CHAR_EVENT *)(msg.body.event.event_ptr))->lead_status
#define CHE_msec(msg)        ((CHAR_EVENT *)(msg.body.event.event_ptr))->msec
#define CHE_second(msg)      ((CHAR_EVENT *)(msg.body.event.event_ptr))->second
#define CHE_minute(msg)      ((CHAR_EVENT *)(msg.body.event.event_ptr))->minute
#define CHE_hour(msg)        ((CHAR_EVENT *)(msg.body.event.event_ptr))->hour
#define CHE_data_char(msg)   ((CHAR_EVENT *)(msg.body.event.event_ptr))->data_char

/*********************************/
/* Character Error Check Event:  */
/*********************************/
typedef struct
{
   INT8  subtype;      /* DTE_ERCK, DCE_ERCK */
   INT8  held_count;   /* how many processes are holding this event        */
   INT16 lead_status;  /* bit-mapped status of up to 45 leads              */
   INT16 msec;         /* 0-999 */
   INT8  second;       /* 0-59 */
   INT8  minute;       /* 0-59 */
   INT8  hour;         /* 0-23 */
   INT8  error_type;   /* GOOD_BCC, BAD_BCC, PARITY_ERROR */
} CHAR_ERCK_EVENT;
```

```
/* macros to access character error check event fields */
#define CH_ERCK_subtype(msg)     ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->subtype
#define CH_ERCK_held_count(msg)  ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->held_count
#define CH_ERCK_lead_status(msg) ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->lead_status
#define CH_ERCK_msec(msg)        ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->msec
#define CH_ERCK_second(msg)      ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->second
#define CH_ERCK_minute(msg)      ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->minute
#define CH_ERCK_hour(msg)        ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))->hour
#define CH_ERCK_error_type(msg) ((CHAR_ERCK_EVENT *)(msg.body.event.event_ptr))-> error_type

/*****************/
/* Frame Event:  */
/*****************/
typedef struct   /* 16 bytes */
{
   INT8  subtype;      /* DTE_FRAME=1,DCE_FRAME=2,SDTE_FRAME=3,SDCE_FRAME=4 */
   INT8  held_count;   /* how many processes are holding this event         */
   INT16 lead_status;  /* bit-mapped status of up to 45 leads               */
   INT16 msec;         /* 0-999 */
   INT8  second;       /* 0-59 */
   INT8  minute;       /* 0-59 */
   INT8  hour;         /* 0-23 */
   INT8  fcs;          /* GOOD=0,ABORT=1,BAD=2 */
   INT16 length;       /* 1-4106 */
   INT8  *frame;       /* ptr to frame contents excl. flags & fcs */
                       /* NOTE: the frame is not NULL terminated  */
} FRAME_EVENT;

/* macros to access frame event fields */
#define FE_subtype(msg)        ((FRAME_EVENT *)(msg.body.event.event_ptr))->subtype
#define FE_held_count(msg)  ((FRAME_EVENT *)(msg.body.event.event_ptr))->held_count
#define FE_lead_status(msg) ((FRAME_EVENT *)(msg.body.event.event_ptr))->lead_status
#define FE_msec(msg)        ((FRAME_EVENT *)(msg.body.event.event_ptr))->msec
#define FE_second(msg)      ((FRAME_EVENT *)(msg.body.event.event_ptr))->second
#define FE_minute(msg)      ((FRAME_EVENT *)(msg.body.event.event_ptr))->minute
#define FE_hour(msg)        ((FRAME_EVENT *)(msg.body.event.event_ptr))->hour
#define FE_fcs(msg)         ((FRAME_EVENT *)(msg.body.event.event_ptr))->fcs
#define FE_length(msg)      ((FRAME_EVENT *)(msg.body.event.event_ptr))->length
#define FE_frame(msg)       ((FRAME_EVENT *)(msg.body.event.event_ptr))->frame
```

```
/****************/
/* Lead Event:  */
/****************/
typedef struct   /* 16 bytes */
{
   INT8  subtype;      /* LEAD_CHANGE=5                                  */
   INT8  held_count;   /* how many processes are holding this event */
   INT16 lead_status;  /* bit-mapped status AFTER the lead change   */
   INT16 msec;         /* 0-999 */
   INT8  second;       /* 0-59 */
   INT8  minute;       /* 0-59 */
   INT8  hour;         /* 0-23 */
   INT8  status;      /* OFF=0,ON=1,NOT_DRIVEN=2 */
   INT16 lead;        /* lead id, defined in leads.include,etc. */
   INT32 unused;      /* needed so that ptr increment works correctly */
} LEAD_EVENT;

/* macros to access lead event fields */
#define LE_subtype(msg)        ((LEAD_EVENT *)(msg.body.event.event_ptr))->subtype
#define LE_held_count(msg)  ((LEAD_EVENT *)(msg.body.event.event_ptr))->held_count
#define LE_lead_status(msg) ((LEAD_EVENT *)(msg.body.event.event_ptr))->lead_status
#define LE_msec(msg)        ((LEAD_EVENT *)(msg.body.event.event_ptr))->msec
#define LE_second(msg)      ((LEAD_EVENT *)(msg.body.event.event_ptr))->second
#define LE_minute(msg)      ((LEAD_EVENT *)(msg.body.event.event_ptr))->minute
#define LE_hour(msg)        ((LEAD_EVENT *)(msg.body.event.event_ptr))->hour
#define LE_status(msg)      ((LEAD_EVENT *)(msg.body.event.event_ptr))->status
#define LE_lead(msg)        ((LEAD_EVENT *)(msg.body.event.event_ptr))->lead

/*******************/
/* Event Structure */
/*******************/
typedef struct
{
   void  *event_ptr;       /* Pointer to either LEAD_EVENT,FRAME_EVENT,    */
                           /* CHAR_EVENT or a CHAR_ERCK_EVENT event        */
                           /* depending on the value of the subtype field  */
   INT8  *data_ptr;        /*beginning of raw data for current level       */
   INT16 length;           /*remaining length of raw data for current level*/
} EVENT;
```

```
/* macros to access the EVENT fields */
#define EV_event_ptr(msg)    (msg.body.event.event_ptr)
#define EV_data_ptr(msg)     (msg.body.event.data_ptr)
#define EV_length(msg)       (msg.body.event.length)

/********************/
/* Timer Structure */
/********************/
typedef struct /* structure used for timer messages          */
{
   INT32 filler[3];
   INT16 timer_number;  /* timer number returned in 'timer_number'*/
                        /* when a timer times out                 */
} TIMER_MESSAGE;

/* macro to access the timer number */
#define TIMER_number(msg)    (msg.body.timer_message.timer_number)

/********************/
/* User  Structure */
/********************/
typedef struct /* structure for user messages */
{
    INT32 user_long;
    INT16 user_int1;
    INT16 user_int2;
    INT16 user_int3;
    INT16 user_int4;
    INT16 user_int5;
} USER_MESSAGE;

/********************/
/* Error Structure */
/********************/
typedef struct /*structure used to hold the message sent */
               /*by error_notification()                  */
{
    char *yelp_msg_ptr;
    INT16 body[5];
} ERROR_MESSAGE;
```

```
/*****************************************************************/
/* Message Structure - this structure is used for all DataCommC  */
/*                     interprocess communication                */
/*****************************************************************/
typedef struct
{
   INT32 Reserved1;        /* reserved        */
   INT32 Reserved2;        /* reserved        */
   INT8  type;             /* message type    */
   INT8  subtype;          /* message subtype */
   union                   /* message body    */
   {
   EVENT          event;
   TIMER_MESSAGE  timer_message;
   USER_MESSAGE   user_message;
   ERROR_MESSAGE  error_message;
   } body;
} MESSAGE;
```

retval.include

This include file contains status and error return code definitions.

```
/* RETURN VALUES FOR LIBRARIES */
#define SUCCESSFUL                0
#define UNSUCCESSFUL             -1

#define ERROR_1                  -1
#define ERROR_2                  -2
#define ERROR_3                  -3
#define ERROR_4                  -4
#define ERROR_5                  -5
#define ERROR_6                  -6
#define ERROR_7                  -7
#define ERROR_8                  -8
#define ERROR_9                  -9
#define ERROR_10                 -10
#define ERROR_100                -100

#define WARNING_1                 1
#define WARNING_2                 2
#define WARNING_3                 3
```

setjmp.include

This include file defines values for `setjmp()` and `longjmp()`.

```
#define   JBUFSIZE  14

typedef long jmp_buf[JBUFSIZE];
```

stdio.include

This include file defines standard I/O values and macros, and data structures for accessing disc files.

```
#ifndef _STDIO_H
#define _STDIO_H
#define fgetc getc
#define fputc putc

#ifndef NULL
#define NULL                    0L
#endif

#define EOF                    -1

#define TRUE                    1
#define FALSE                   0

#define STDINSIZE              80
#define BUFSIZ               1024
#define MAXSTREAM              20

#define _BUSY                0x01
#define _ALLBUF              0x02
#define _DIRTY               0x04
#define _EOF                 0x08
#define _IOERR               0x10
#define _TEMP                0x20  /* temporary file (delete on close) */

typedef struct {
    char *_bp;          /* current position in buffer */
    char *_bend;        /* last character in buffer + 1 */
    char *_buff;        /* address of buffer */
    char _flags;        /* open mode, etc. */
    char _unit;         /* token returned by open */
    char _bytbuf;       /* single byte buffer for unbuffer streams */
    short _buflen;      /* length of buffer */
    char *_tmpname;     /* name of file for temporaries */
} FILE;
```

```
extern FILE Cbuffs[];
FILE *fopen();
long ftell();
long lseek();

#define stdin         (&Cbuffs[0])
#define stdout        (&Cbuffs[1])
#define stderr        (&Cbuffs[2])
#define getchar()     agetc(stdin)
#define putchar(c)    aputc(c, stdout)
#define feof(fp)      (((fp)->_flags&_EOF)!=0)
#define ferror(fp)    (((fp)->_flags&_IOERR)!=0)
#define clearerr(fp) ((fp)->_flags &= ~(_IOERR|_EOF))
#define fileno(fp)    ((fp)->_unit)
#define fflush(fp)    flsh_(fp,-1)
#define rewind(fp)    fseek(fp, 0L, 0)
#define P_tmpdir      ""
#define L_tmpnam      40

#endif

/* origin type for fseek and lseek */
#define SEEK_SET                  0
#define SEEK_CUR                  1
#define SEEK_END                  2
```

stdlib.include

This include file contains function definitions for the standard library.

```
double atof();
int  atoi();
long atol();
char *calloc();
void free();
char *itoa();
char *ltoa();
char *ftoa();
char *malloc();
```

string.include

This include file contains declarations for string manipulation functions.

```
char *strcat();
char *strchr();
char *strcpy();
char *strncat();
char *strncpy();
char *strrchr();
char *strsave();
char *strstr();
char *strtrm();
```

system.include

This include file defines values for system (PSOS) status and error codes.

```
#define DEFAULT_PRIORITY              100

/* Spawn error codes          */
#define CANT_OPEN_PROGRAM_FILE         -1
#define NO_MEMORY_FOR_CHILD            -2
#define INVALID_PROGRAM_FILE_FORMAT    -3
#define NO_QUEUES_AVAILABLE           -21
#define CANT_NOTIFY_PROCESS_MGR       -22
#define TOO_MANY_PROCESSES            -41
#define NO_MEMORY_FOR_STACK           -42
#define STACK_TOO_SMALL               -43

/* the following symbols are system variable and if modified by a user pgm.  */
/* unpredictable results will occur!!! */

extern long _process_QID,   /*QID of present process            */
            _parent_QID,    /*parent's QID                       */
            _code_segment;  /*beginning of process code segment */
```

time.include

This include file defines the data structure used in the real-time clock access functions.

```
/* tm structure used by get_time() and set_time() */
struct tm
   {
   int tm_msecs;
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_day;
   int tm_mon;
   int tm_year;
   };
```

video.include

This include file contains values for the video I/O functions.

```
/*** constants used by set_attribute() and get_attribute() ***/

/* ASCII character attributes */
#define NORMAL                  0x0200
#define INVERSE                 0x0600
#define HALF_BRIGHT             0x0A00
#define BLINK                   0x1200

/* HEX character attributes */
#define HEX_NORMAL              0x0000
#define HEX_INVERSE             0x0400
#define HEX_HALF_BRIGHT         0x0800
#define HEX_BLINK               0x1000
```

```
/* EBCDIC character attributes */
#define EBCDIC_NORMAL       0x0100
#define EBCDIC_INVERSE      0x0500
#define EBCDIC_HALF_BRIGHT 0x0900
#define EBCDIC_BLINK        0x1100

/* attribute for graphics characters */
#define SPECIAL          0x0300

#define UNDERLINE        0x2000

/*** constants used by set_disp_bank() ***/

/* Display Bank Constants */
#define DISP_BANK0              0x000
#define DISP_BANK1              0x100
#define DISP_BANK2              0x200
#define DISP_BANK3              0x300

/*** constants used by set_screen_mode() ***/
#define ASCII_DISPLAY       0
#define TRANSPARENT_DISPLAY 1
```

A

Video Character Sets

The DataCommC Programming Language gives you access to the HP 4954A Protocol Analyzer's ASCII, Hex and EBCDIC video character sets, as well as four additional display banks that include graphics and Katakana characters. The character sets may be selected with the `set_attribute` function, and if the `SPECIAL` attribute is chosen, one of the four graphics display banks may be selected using `set_disp_bank()`.

The individual characters contained in the character sets and display banks are shown in the tables on the following pages.

Graphics Display Characters

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
128	80	⎺	⎺		⎺
129	81	⎸	⎸	⎺	⎸
130	82	⎾	⎾	⎺	⎾
131	83	⎿	⎿	⎺	⎿
132	84	⏌	⏌	⎺	⏌
133	85	⏋	⏋	⎺	⏋
134	86	⏊	⏊	⎺	⏊
135	87	⏉	⏉	⎺	⏉
136	88	⏈	⏈	⎺	⏈
137	89	⏇	⏇	⎺	⏇
138	8A	⎻	⎻	⎺	⎻
139	8B	⎼	⎼	⎺	⎼
140	8C	⎽	⎽	⎺	⎽
141	8D	⏌	⏋	⎺	⎺
142	8E	⏊	⏉	⎺	⎼
143	8F	⏈	⏇	⎺	⎾
144	90	⏋	⏊	⎺	⏋
145	91	⏉	⏈	⎺	⏇

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
146	92	⏇	⎺	⎺	⏇
147	93	⏇	⎽	⏋	⏇
148	94	⏇	⏇	⏋	⏇
149	95	⏇	⏇	⏋	⏇
150	96	⏇	⏇	⏋	⏇
151	97	⏇	⏇	⏋	⏇
152	98	⏇	⏇	⏋	⏇
153	99	⏇	⏇	⏋	⏇
154	9A	⏇	⏇	⏋	⏇
155	9B	⏇	⏊	⏋	⏇
156	9C	⏇	⏈	⏋	⏇
157	9D	⏇	⎺	⏋	⏇
158	9E	⏇	⏉	⏋	⏇
159	9F	⏇	⏇	⏋	⏇
160	A0	⏇	⏇		⏇
161	A1	⏇	⏇	⎽	⏇
162	A2	⏇	⏇	⎽	⏇
163	A3	⏇	⏇	⎽	⏇

Graphics Display Characters (cont'd)

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
164	A4				
165	A5				
166	A6				
167	A7				
168	A8				
169	A9				
170	AA				
171	AB				
172	AC				
173	AD				
174	AE				
175	AF				
176	B0				
177	B1				
178	B2				
179	B3				
180	B4				
181	B5				

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
182	B6				
183	B7				
184	B8				
185	B9				
186	BA				
187	BB				
188	BC				
189	BD				
190	BE				
191	BF				
192	C0				
193	C1				
194	C2				
195	C3				
196	C4				
197	C5				
198	C6				
199	C7				

Graphics Display Characters (cont'd)

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
200	C8				
201	C9				
202	CA				
203	CB				
204	CC				
205	CD	ュ	⸤		
206	CE	ョ			
207	CF	ッ			
208	D0	¥			
209	D1	ア		—	
210	D2	イ		‾	
211	D3	ウ		‾	
212	D4	エ		‾	
213	D5	オ		‾	
214	D6	カ		‾	
215	D7	キ		‾	
216	D8	ク		‾	
217	D9	ケ		‾	

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
218	DA	コ		_	
219	DB	サ		_	
220	DC	シ		_	
221	DD	ス		_	
222	DE	セ		_	
223	DF	ソ		_	
224	E0	タ			
225	E1	チ		_	
226	E2	ツ		_	
227	E3	テ		_	
228	E4	ト		_	
229	E5	ナ		_	
230	E6	ニ		_	
231	E7	ヌ		_	
232	E8	ネ		_	
233	E9	ノ		_	
234	EA	ハ		_	
235	EB	ヒ		_	

Graphics Display Characters (cont'd)

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
164	A4				
165	A5				
166	A6				
167	A7				
168	A8				
169	A9				
170	AA				
171	AB				
172	AC				
173	AD				
174	AE				
175	AF				
176	B0				
177	B1				
178	B2				
179	B3				
180	B4				
181	B5				

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
182	B6				
183	B7				
184	B8				
185	B9				
186	BA				
187	BB				
188	BC				
189	BD				
190	BE				
191	BF				
192	C0				
193	C1				
194	C2				
195	C3				
196	C4				
197	C5				
198	C6				
199	C7				

Graphics Display Characters (cont'd)

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
200	C8				
201	C9				
202	CA				
203	CB				
204	CC				
205	CD				
206	CE				
207	CF				
208	D0				
209	D1				
210	D2				
211	D3				
212	D4				
213	D5				
214	D6				
215	D7				
216	D8				
217	D9				

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
218	DA				
219	DB				
220	DC				
221	DD				
222	DE				
223	DF				
224	E0				
225	E1				
226	E2				
227	E3				
228	E4				
229	E5				
230	E6				
231	E7				
232	E8				
233	E9				
234	EA				
235	EB				

Graphics Display Characters (cont'd)

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
236	EC	ｱ			
237	ED	ｲ			
238	EE	ｳ			
239	EF	ｴ			
240	F0			■	
241	F1			|	
242	F2			·	
243	F3			–	
244	F4			┐	
245	F5			┬	
246	F6	·	·	┐	·
247	F7			-	
248	F8	■	■	|	■
249	F9			┐	
250	FA	─	─	©	─
251	FB	|	|		|
252	FC	┌	┌	□	┌
253	FD	└	└		└

Dec	Hex	Bank 0	Bank 1	Bank 2	Bank 3
254	FE	┘	┘		┘
255	FF	┐	┐	||	┐

ASCII, Hex and EBCDIC Display Characters

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
0	00	NU	0 0	NU
1	01	SH	0 1	SH
2	02	SX	0 2	SX
3	03	EX	0 3	EX
4	04	ET	0 4	PF
5	05	EQ	0 5	HT
6	06	AK	0 6	LC
7	07	BL	0 7	
8	08	BS	0 8	0 8
9	09	HT	0 9	RF
10	0A	LF	0 A	SM
11	0B	VT	0 B	VT
12	0C	FF	0 C	FF
13	0D	CR	0 D	CR
14	0E	SO	0 E	SO
15	0F	SI	0 F	SI

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
16	10	DL	1 0	DL
17	11	D1	1 1	D1
18	12	D2	1 2	D2
19	13	D3	1 3	D3
20	14	D4	1 4	PE
21	15	NK	1 5	NL
22	16	SY	1 6	BS
23	17	EB	1 7	IL
24	18	CN	1 8	CN
25	19	EM	1 9	EM
26	1A	SB	1 A	CC
27	1B	EC	1 B	C1
28	1C	FS	1 C	FS
29	1D	GS	1 D	GS
30	1E	RS	1 E	RS
31	1F	US	1 F	US

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
32	20		2_0	4_0
33	21	!	2_1	5_A
34	22	"	2_2	7_F
35	23	#	2_3	7_B
36	24	$	2_4	5_B
37	25	%	2_5	6_C
38	26	&	2_6	5_0
39	27	'	2_7	7_D
40	28	(	2_8	4_D
41	29	)	2_9	5_D
42	2A	*	2_A	5_C
43	2B	+	2_B	4_E
44	2C	,	2_C	6_B
45	2D	-	2_D	6_0
46	2E	.	2_E	4_B
47	2F	/	2_F	6_1

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
48	30	0	3_0	F_0
49	31	1	3_1	F_1
50	32	2	3_2	F_2
51	33	3	3_3	F_3
52	34	4	3_4	F_4
53	35	5	3_5	F_5
54	36	6	3_6	F_6
55	37	7	3_7	F_7
56	38	8	3_8	F_8
57	39	9	3_9	F_9
58	3A	:	3_A	7_A
59	3B	;	3_B	5_E
60	3C	<	3_C	4_C
61	3D	=	3_D	7_E
62	3E	>	3_E	6_E
63	3F	?	3_F	6_F

ASCII, Hex and EBCDIC Display Characters (cont'd)

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
64	40	@	[illegible]	
65	41	A	[illegible]	[illegible]
66	42	B	[illegible]	[illegible]
67	43	C	[illegible]	[illegible]
68	44	D	[illegible]	[illegible]
69	45	E	[illegible]	[illegible]
70	46	F	[illegible]	[illegible]
71	47	G	[illegible]	[illegible]
72	48	H	[illegible]	[illegible]
73	49	I	[illegible]	[illegible]
74	4A	J	[illegible]	¢
75	4B	K	[illegible]	.
76	4C	L	[illegible]	<
77	4D	M	[illegible]	(
78	4E	N	[illegible]	+
79	4F	O	[illegible]	|

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
80	50	P	[illegible]	&
81	51	Q	[illegible]	[illegible]
82	52	R	[illegible]	[illegible]
83	53	S	[illegible]	[illegible]
84	54	T	[illegible]	[illegible]
85	55	U	[illegible]	[illegible]
86	56	V	[illegible]	[illegible]
87	57	W	[illegible]	[illegible]
88	58	X	[illegible]	[illegible]
89	59	Y	[illegible]	[illegible]
90	5A	Z	[illegible]	!
91	5B	[	[illegible]	$
92	5C	\	[illegible]	*
93	5D	]	[illegible]	)
94	5E	^	[illegible]	;
95	5F	–	[illegible]	¬

ASCII, Hex and EBCDIC Display Characters (cont'd)

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
96	60	`	6/0	–
97	61	a	6/1	/
98	62	b	6/2	6/2
99	63	c	6/3	6/3
100	64	d	6/4	6/4
101	65	e	6/5	6/5
102	66	f	6/6	6/6
103	67	g	6/7	6/7
104	68	h	6/8	6/8
105	69	i	6/9	6/9
106	6A	j	6/A	¦
107	6B	k	6/B	,
108	6C	l		%
109	6D	m	6/D	_
110	6E	n	6/E	>
111	6F	o	6/F	?

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
112	70	p	7/0	7/0
113	71	q	7/1	7/1
114	72	r	7/2	7/2
115	73	s	7/3	7/3
116	74	t	7/4	7/4
117	75	u	7/5	7/5
118	76	v	7/6	7/6
119	77	w	7/7	7/7
120	78	x	7/8	7/8
121	79	y	7/9	`
122	7A	z	7/A	:
123	7B	{	7/B	#
124	7C	¦	7/C	@
125	7D	}	7/D	'
126	7E	~	7/E	=
127	7F	▨	7/F	"

ASCII, Hex and EBCDIC Display Characters (cont'd)

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
128	80		$_0$	$_0$
129	81		$_1$	a
130	82		$_2$	b
131	83		$_3$	c
132	84		$_4$	d
133	85		$_5$	e
134	86		$_6$	f
135	87		$_7$	g
136	88		$_8$	h
137	89		$_9$	i
138	8A		$_A$	$_A$
139	8B		$_B$	$_B$
140	8C		$_C$	$_C$
141	8D		$_D$	$_D$
142	8E		$_E$	$_E$
143	8F		$_F$	$_F$

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
144	90		$_0$	$_0$
145	91		$_1$	j
146	92		$_2$	k
147	93		$_3$	l
148	94		$_4$	m
149	95		$_5$	n
150	96		$_6$	o
151	97		$_7$	p
152	98		$_8$	q
153	99		$_9$	r
154	9A		$_A$	$_A$
155	9B		$_B$	$_B$
156	9C		$_C$	$_C$
157	9D		$_D$	$_D$
158	9E		$_E$	$_E$
159	9F		$_F$	$_F$

ASCII, Hex and EBCDIC Display Characters (cont'd)

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
160	A0			
161	A1			~
162	A2			s
163	A3			t
164	A4			u
165	A5			v
166	A6			w
167	A7			x
168	A8			y
169	A9			z
170	AA			
171	AB			
172	AC			
173	AD			
174	AE			
175	AF			

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
176	B0			
177	B1			
178	B2			
179	B3			
180	B4			
181	B5			
182	B6			
183	B7			
184	B8			
185	B9			
186	BA			
187	BB			
188	BC			
189	BD			
190	BE			
191	BF			

ASCII, Hex and EBCDIC Display Characters (cont'd)

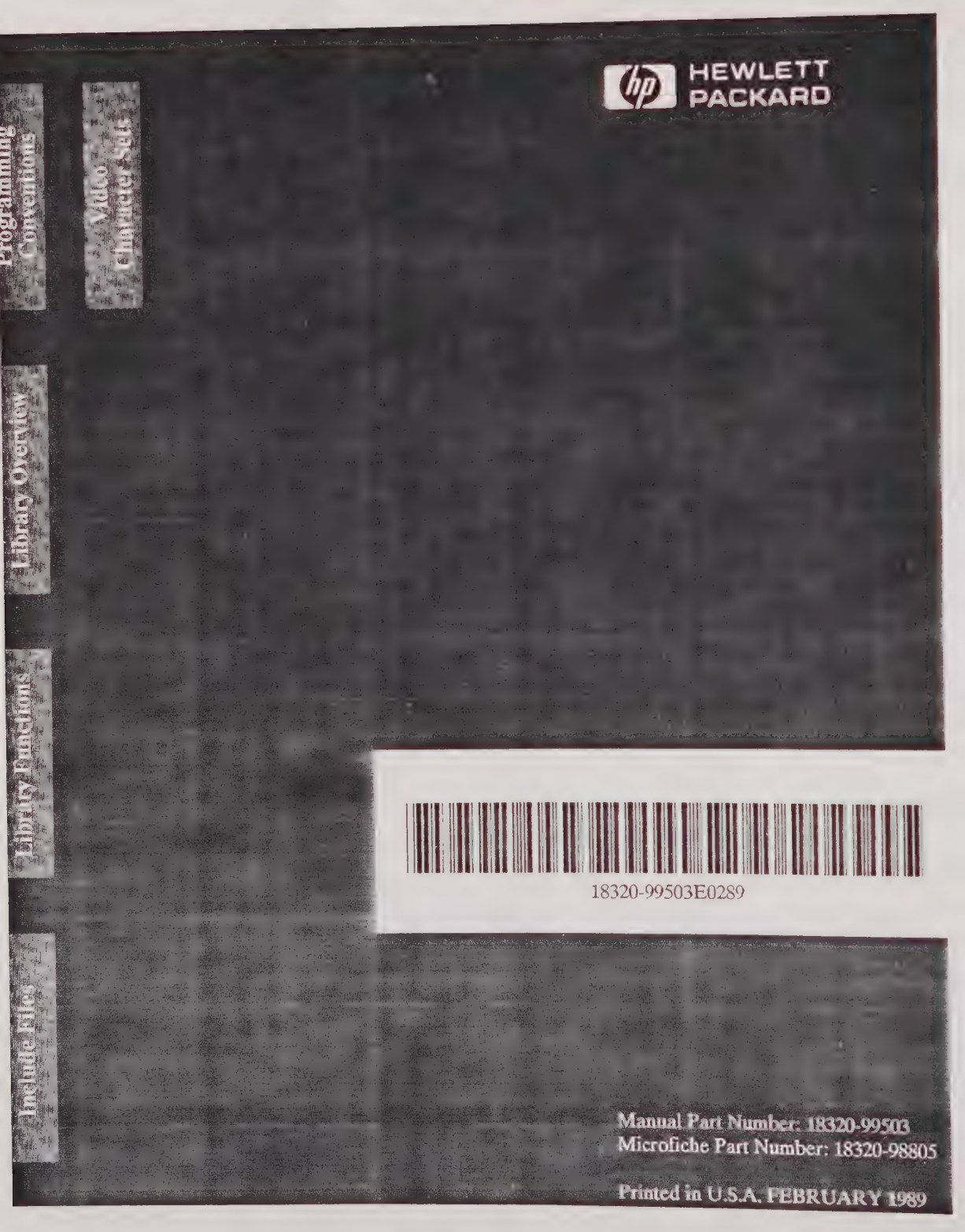
Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
192	C0			{
193	C1			A
194	C2			B
195	C3			C
196	C4			D
197	C5			E
198	C6			F
199	C7			G
200	C8			H
201	C9			I
202	CA			
203	CB			
204	CC			
205	CD			
206	CE			
207	CF			

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
208	D0			}
209	D1			J
210	D2			K
211	D3			L
212	D4			M
213	D5			N
214	D6			O
215	D7			P
216	D8			Q
217	D9			R
218	DA			
219	DB			
220	DC			
221	DD			
222	DE			
223	DF			

ASCII, Hex and EBCDIC Display Characters (cont'd)

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
224	E0		E0	\
225	E1		E1	E1
226	E2		E2	S
227	E3		E3	T
228	E4		E4	U
229	E5		E5	V
230	E6		E6	W
231	E7		E7	X
232	E8		E8	Y
233	E9		E9	Z
234	EA		EA	EA
235	EB		EB	EB
236	EC		EC	H
237	ED		ED	ED
238	EE		EE	EE
239	EF		EF	EF

Value		Display Character		
Dec	Hex	ASCII	Hex	EBCDIC
240	F0		F0	0
241	F1		F1	1
242	F2		F2	2
243	F3		F3	3
244	F4		F4	4
245	F5		F5	5
246	F6		F6	6
247	F7		F7	7
248	F8		F8	8
249	F9		F9	9
250	FA		FA	I
251	FB		FB	FB
252	FC		FC	FC
253	FD		FD	FD
254	FE		FE	FE
255	FF		FF	FF


HEWLETT PACKARD
Programming Conventions
Video Character Set
Library Overview
Library Functions
Include Files
18320-99503E0289
Manual Part Number: 18320-99503
Microfiche Part Number: 18320-98805
Printed in U.S.A. FEBRUARY 1989

HEWLETT PACKARD

Reorder No. or
Manual Part No.
18320-99503-E0289

9282-1161
Binder Only Part No
Made in U.S.A.